

Robert Meersman
Zahir Tari et al. (Eds.)

LNCS 3760

On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE

OTM Confederated International Conferences
CoopIS, DOA, and ODBASE 2005
Agia Napa, Cyprus, October 2005
Proceedings, Part I

1
Part I



DOA



Springer

Commenced Publication in 1973

Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

New York University, NY, USA

Doug Tygar

University of California, Berkeley, CA, USA

Moshe Y. Vardi

Rice University, Houston, TX, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Robert Meersman Zahir Tari
Mohand-Saïd Hacid John Mylopoulos
Barbara Pernici Ozalp Babaoglu
H.-Arno Jacobsen Joseph Loyall
Michael Kifer Stefano Spaccapietra (Eds.)

On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE

OTM Confederated International Conferences
CoopIS, DOA, and ODBASE 2005
Agia Napa, Cyprus, October 31 – November 4, 2005
Proceedings, Part I

Volume Editors

Robert Meersman

Vrije Universiteit Brussel , STAR Lab
Pleinlaan 2, Bldg G/10, 1050 Brussels, Belgium
E-mail: meersman@vub.ac.be

Zahir Tari

RMIT University, School of Computer Science and Information Technology
City Campus, GPO Box 2476 V, Melbourne, Victoria 3001, Australia
E-mail: zahirt@cs.rmit.edu.au

Library of Congress Control Number: 2005934471

CR Subject Classification (1998): H.2, H.3, H.4, C.2, H.5, I.2, D.2.12, K.4

ISSN 0302-9743

ISBN-10 3-540-29736-7 Springer Berlin Heidelberg New York

ISBN-13 978-3-540-29736-9 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springeronline.com

© Springer-Verlag Berlin Heidelberg 2005
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11575771 06/3142 5 4 3 2 1 0

CoopIS
Mohand-Saïd Hacid
John Mylopoulos
Barbara Pernici

DOA
Ozalp Babaoglu
H.-Arno Jacobsen
Joseph Loyall

ODBASE
Michael Kifer
Stefano Spaccapietra

OTM 2005 General Co-chairs' Message

The General Chairs of OnTheMove 2005, Agia Napa, Cyprus, are happy to observe that the conference series that was started in Irvine, California in 2002, and continued in Catania, Sicily, in 2003, and in the same location in Cyprus last year, clearly supports a scientific concept that attracts a representative selection of today's worldwide research in distributed, heterogeneous and autonomous yet meaningfully collaborative computing, of which the Internet and the WWW are its prime epitomes.

Indeed, as such large, complex and networked intelligent information systems become the focus and norm for computing, it is clear that there is an acute need to address and discuss in an integrated forum the implied software and system issues as well as methodological, theoretical and application issues. As we all know, email, the Internet, and even video conferences are not sufficient for effective and efficient scientific exchange. This is why the OnTheMove (OTM) Federated Conferences series has been created to cover the increasingly wide yet closely connected range of fundamental technologies such as data and Web Semantics, distributed objects, Web services, databases, information systems, workflow, cooperation, ubiquity, interoperability, and mobility. OTM aspires to be a primary scientific meeting place where all aspects for the development of internet- and intranet-based systems in organizations and for e-business are discussed in a scientifically motivated way. This fourth 2005 edition of the OTM Federated Conferences therefore again provides an opportunity for researchers and practitioners to understand and publish these developments within their individual as well as within their broader contexts.

The backbone of OTM is formed by the co-location of three related, complementary and successful main conference series: DOA (Distributed Objects and Applications, since 1999), covering the relevant infrastructure-enabling technologies; ODBASE (Ontologies, DataBases and Applications of SEMantics, since 2002), covering Web semantics, XML databases and ontologies; and CoopIS (Co-operative Information Systems, since 1993), covering the application of these technologies in an enterprise context through, e.g., workflow systems and knowledge management. Each of these three conferences encourages researchers to treat their respective topics within a framework that incorporates jointly (a) theory, (b) conceptual design and development, and (c) applications, in particular case studies and industrial solutions.

Following and expanding the model created in 2003, we again solicited and selected quality workshop proposals to complement the more "archival" nature of the main conferences with research results in a number of selected and more "avant garde" areas related to the general topic of distributed computing. For instance, the so-called Semantic Web has given rise to several novel research areas combining linguistics, information systems technology, and artificial intelligence,

such as the modeling of (legal) regulatory systems and the ubiquitous nature of their usage. We were glad to see that in 2005 under the inspired leadership of Dr. Pilar Herrero, several of earlier successful workshops re-emerged with a second or even third edition (notably WOSE, MIOS-INTEROP and GADA), and that 5 new workshops could be hosted and successfully organized by their respective proposers: AWeSOMe, SWWS, CAMS, ORM and SeBGIS. We know that as before, their audiences will mutually productively mingle with those of the main conferences, as is already visible from the overlap in authors!

A special mention for 2005 is again due for the second and enlarged edition of the highly successful Doctoral Symposium Workshop. Its 2005 Chairs, Dr. Antonia Albani, Dr. Peter Spyns, and Dr. Johannes Maria Zaha, three young and active post-doc researchers defined an original set-up and interactive formula to bring PhD students together: they call them to submit their research proposals for selection; the resulting submissions and their approaches are presented by the students in front of a wider audience at the conference, where they are then independently analyzed and discussed by a panel of senior professors (this year they were Domenico Beneventano, Jaime Delgado, Jan Dietz, and Werner Nutt). These successful students also get free access to “all” other parts of the OTM program, and only pay a minimal fee for the Doctoral Symposium itself (in fact their attendance is largely sponsored by the other participants!). The OTM organizers expect to further expand this model in future editions of the conferences and so draw an audience of young researchers into the OTM forum.

All three main conferences and the associated workshops share the distributed aspects of modern computing systems, and the resulting application-pull created by the Internet and the so-called Semantic Web. For DOA 2005, the primary emphasis stayed on the distributed object infrastructure; for ODBASE 2005, it became the knowledge bases and methods required for enabling the use of formal semantics, and for CoopIS 2005, the topic was the interaction of such technologies and methods with management issues, such as occur in networked organizations. These subject areas naturally overlap and many submissions in fact also treat an envisaged mutual impact among them. As for the earlier editions, the organizers wanted to stimulate this cross-pollination by a “shared” program of famous keynote speakers: this year we got no less than Erich Neuhold (Emeritus, Fraunhofer/IPS), Stefano Ceri (Politecnico di Milano), Doug Schmidt (Vanderbilt University), and V.S. Subrahmanian (University of Maryland)! We also encouraged multiple event attendance by providing “all” authors, also those of workshop papers, with free access or discounts to one other conference or workshop of their choice.

We received a total of 360 submissions for the three main conferences and a whopping 268 (compared to the 170 in 2004!) in total for the workshops. Not only can we therefore again claim success in attracting an increasingly representative volume of scientific papers, but such a harvest of course allows the program committees to compose a higher quality cross-section of current research in the areas covered by OTM. In fact, in spite of the larger number of submissions, the Program Chairs of each of the three main conferences decided to accept only

approximately the same number of papers for presentation and publication as in 2003 and 2004 (i.e., average 1 paper out of 4 submitted, not counting posters). For the workshops, the acceptance rate varies but was much stricter than before, about 1 in 2-3, to almost 1 in 4 for GADA and MIOS. Also for this reason, we continue to separate the proceedings in two books with their own titles, with the main proceedings in two volumes, and we are grateful to Springer for their suggestions and collaboration in producing these books and CD-ROMs. The reviewing process by the respective program committees as usual was performed very professionally and each paper in the main conferences was reviewed by at least three referees, with email discussions in the case of strongly diverging evaluations. It may be worthwhile to emphasize that it is an explicit OTM policy that all conference program committees and chairs make their selections completely autonomously from the OTM organization itself. Continuing a costly but nice tradition, the OTM Federated Event organizers decided again to make all proceedings available as books and/or CD-ROMs to all participants of conferences and workshops, independently of one's registration.

The General Chairs are once more especially grateful to all the many people directly or indirectly involved in the set-up of these federated conferences and in doing so made this a success. Few people realize what a large number of individuals have to be involved, and what a huge amount of work, and sometimes risk, the organization of an event like OTM entails. Apart from the persons in their roles mentioned above, we therefore in particular wish to thank our 8 main conference PC Co-chairs (DOA 2005: Ozalp Babaoglu, Arno Jacobsen, Joe Loyall; ODBASE 2005: Michael Kifer, Stefano Spaccapietra; CoopIS 2005: Mohand-Said Hacid, John Mylopoulos, Barbara Pernici), and our 26 workshop PC Co-chairs (Antonia Albani, Lora Aroyo, George Buchanan, Lawrence Cavendon, Jan Dietz, Tharam Dillon, Erik Duval, Ling Feng, Aldo Gangemi, Annika Hinze, Mustafa Jarrar, Terry Halpin, Pilar Herrero, Jan Humble, David Martin, Gonzalo Médez, Aldo de Moor, Hervé Panetto, María S. Pérez, Víctor Robles, Monica Scannapieco, Peter Spyns, Emmanuel Stefanakis, Klaus Turowski, Esteban Zimányi). All, together with their many PCs, members did a superb and professional job in selecting the best papers from the large harvest of submissions. We also thank Laura Bright, our excellent Publicity Chair for the second year in a row, our Conference Secretariat staff and Technical Support Daniel Meersman and Jan Demey, and last but not least our hyperactive Publications Chair and loyal collaborator of many years, Kwong Yuen Lai.

The General Chairs gratefully acknowledge the logistic support and facilities they enjoy from their respective institutions, Vrije Universiteit Brussel (VUB) and RMIT University, Melbourne.

We do hope that the results of this federated scientific enterprise contribute to your research and your place in the scientific network... We look forward to seeing you again at next year's edition!

August 2005

Robert Meersman, Vrije Universiteit Brussel, Belgium
Zahir Tari, RMIT University, Australia
(General Co-chairs, OnTheMove 2005)

Organization Committee

The OTM (On The Move) 2005 Federated Conferences, which involve CoopIS (Cooperative Information Systems), DOA (Distributed Objects and Applications) and ODBASE (Ontologies, Databases and Applications of Semantics), are proudly supported by RMIT University (School of Computer Science, Information Technology) and Vrije Universiteit Brussel (Department of Computer Science) and Interop.

Executive Committee

OTM 2005 General Co-chairs	Robert Meersman (Vrije Universiteit Brussel, Belgium) and Zahir Tari (RMIT University, Australia)
CoopIS 2005 PC Co-chairs	Mohand-Said Hacid (Université Claude Bernard Lyon I), John Mylopoulos (University of Toronto), and Barbara Pernici (Politecnico di Milano)
DOA 2005 PC Co-chairs	Ozalp Babaoglu (University of Bologna), Arno Jacobsen (University of Toronto), and Joe Loyall (BBN Technologies)
ODBASE 2005 PC Co-chairs	Michael Kifer (Stony Brook University) and Stefano Spaccapietra (Swiss Federal Institute of Technology at Lausanne)
Publication Co-chairs	Kwong Yuen Lai (RMIT University, Australia) and Peter Dimopoulos (RMIT University, Australia)
Organizing Chair	Skevos Evripidou (University of Cyprus, Cyprus)
Publicity Chair	Laura Bright (Oregon Graduate Institute, Oregon, USA)

CoopIS 2005 Program Committee

Wil van der Aalst	Salima Benbernou
Bernd Amann	Djamal Benslimane
Lefteris Angelis	Elisa Bertino
Naveen Ashish	Athman Bouguettaya
Alistair Barros	Mokrane Bouzeghoub
Zohra Bellahsene	Christoph Bussler
Boualem Benatallah	Barbara Carminati

Fabio Casati	Maristella Matera
Malu Castellanos	Massimo Mecella
Barbara Catania	Michael zur Muehlen
Henning Christiansen	Werner Nutt
Bin Cui	Andreas Oberweis
Umesh Dayal	Jean-Marc Petit
Alex Delis	Evaggelia Pitoura
Drew Devereux	Alessandro Provetti
Susanna Donatelli	Zbigniew W. Ras
Marlon Dumas	Manfred Reichert
Schahram Dustdar	Tore Risch
Johann Eder	Marie-Christine Rousset
Rik Eshuis	Kai-Uwe Sattler
Opher Etzion	Monica Scannapieco
Elena Ferrari	Ralf Schenkel
Avigdor Gal	Antonio Si
Paul Grefen	Farouk Toumani
Manfred Hauswirth	Susan Urban
Geert-Jan Houben	Athena Vakali
Michael Huhns	Mathias Weske
Paul Johannesson	Kyu-Young Whang
Latifur Khan	Jian Yang
Manolis Koubarakis	Ming Yung
Akhil Kumar	Arkady Zaslavsky
Winfried Lamersdorf	Leon Zhao
Steven Laufmann	Roger Zimmermann
Qing Li	

CoopIS 2005 Additional Reviewers

Rong Liu	Camelia Constantin
Jianrui Wang	Florian Rosenberg
Agnieszka Dardzinska	Benjamin Schmit
Samuil Angelov	Wil van der Aalst
Yigal Hoffner	Ana Karla Alves de Medeiros
Sven Till	Christian Guenther
Jochem Vonk	Eric Verbeek
Stratos Idreos	Aviv Segev
Christos Tryfonopoulos	Mati Golani
Harald Meyer	Ami Eyal
Hagen Overdick	Daniela Berardi
Hilmar Schuschel	Fabio De Rosa
Guido Laures	Woong-Kee Loh
Frank Puhlmann	Jae-Gil Lee

Horst Pichler	Etienne Canaud
Marek Lehmann	Dickson Chiu
Renate Motschnig	Xiang Li
Diego Milano	Zhe Shan
Xumin Liu	Elvis Leung
Qi Yu	Jing Chen
Zaki Malik	Jingshan Huang
Xu Yang	Armin Haller
George Zheng	Kostas Stefanidis
Florian Daniel	Nikos Nikolaidis
Federico Michele Facca	Mick Kerrigan
Jialie Shen	Massimo Marchi
Min Qin	Brahmananda Sapkota
Hong Zhu	Hamid Reza Motahari
Wei-Shinn Ku	Julien Ponge
Leslie S. Liu	Halvard Skogsrud
Bart Orriens	Aixin Sun
James Pruyne	Quan Zheng Sheng
George Pallis	Guy Gouardéres
Vasiliki Koutsonikola	Mar Roantree
Konstantina Stoupa	Pierre Pompidor
Theodosios Theodosiou	Ela Hunt
Sarita Bassil	Anna Cinzia Squicciarini
Fabien DeMarchi	

ODBASE 2005 Program Committee

Juergen Angele	Johann Eder
Alessandro Artale	Tim Finin
Mira Balaban	Enrico Franconi
Denilson Barbosa	Fausto Giunchiglia
Daniela Berardi	Mohand Said Hacid
Sonia Bergamaschi	Jeff Heflin
Abraham Bernstein	Ian Horrocks
Leo Bertossi	Arantza Illarramendi
Harold Boley	Mustafa Jarrar
Alex Borgida	Christopher Jones
Christoph Bussler	Vipul Kashyap
Jos de Bruijn	Larry Kerschberg
Gilberto Câmara	Roger (Buzz) King
Marco Antonio Casanova	Werner Kuhn
Kajal Claypool	Georg Lausen
Mariano Consens	Bertram Ludaescher
Isabel Cruz	Sanjay Madria
Rainer Eckstein	Murali Mani

Leo Mark	Amit Sheth
Wolfgang May	Michael Sintek
Michele Missikoff	Naveen Srinivasan
Boris Motik	Steffen Staab
Saikat Mukherjee	Jianwen Su
Moira Norrie	York Sure
Maria Orlowska	David Toman
Yue Pan	Christophides Vassilis
Christine Parent	Holger Wache
Torben Bach Pedersen	Gerd Wagner
Axel Polleres	Guizhen Yang
Louiqa Raschid	Esteban Zimanyi
Monica Scannapieco	

ODBASE 2005 Additional Reviewers

Alex Binun	Karl Wiggisser
David Boaz	Christian Koncilia
Lior Limonad	Diego Milano
Azzam Marii	Dimitris Kotzinos
Steffen Lamparter	Ansgar Bernardi
Johanna Voelker	Malte Kiesel
Peter Haase	Ludger van Elst
Denny Vrandecic	Hans Trost
Carsten Saathoff	Adrian Giurca
Kalyan Ayloo	Sergey Lukichev
Huiyong Xiao	Michael Lutz
Jesús Bermúdez	Fabio Machado Porto
Alfredo Goñi	Aida Boukottaya
Sergio Ilarri	Matthew Moran
Birte Glimm	Roberta Benassi
Lei Li	Domenico Beneventano
Jeff Pan	Stefania Bruschi
Evgeny Zolin	Francesco Guerra
Francesco Taglino	Mirko Orsini
Antonio De Nicola	James Scicluna
Federica Schiappelli	Cristina Feier
Fulvio D'Antonio	

DOA 2005 Program Committee

Cristiana Amza	Guruduth Banavar
Matthias Anlauff	Alberto Bartoli
Mark Baker	Judith Bishop

Gordon Blair	Keith Moore
Alex Buchmann	Francois Pacull
Harold Carr	Simon Patarin
Michel Chaudron	Joao Pereira
Shing-Chi Cheung	Rajendra Raj
Geoff Coulson	Andry Rakotonirainy
Francisco "Paco" Curbela	Luis Rodrigues
Wolfgang Emmerich	Isabelle Rouvellou
Patrick Eugster	Rick Schantz
Pascal Felber	Heinz-W. Schmidt
Kurt Geihs	Douglas Schmidt
Jeff Gray	Richard Soley
Mohand-Said Hacid	Michael Stal
Rebecca Isaacs	Jean-Bernard Stefani
Mehdi Jazayeri	Stefan Tai
Bettina Kemme	Hong Va Leong
Fabio Kon	Maarten van Steen
Doug Lea	Steve Vinoski
Peter Loehr	Norbert Voelker
Frank Manola	Andrew Watson
Philip McKinley	Doug Wells

DOA 2005 Additional Reviewers

Jochen Fromm	Vinod Muthusamy
Steffen Bleul	Arlindo Flávio da Concei��o
Roland Reichle	Raphael Camargo
Thomas Weise	David Cyrluk
An Chunyan	Klaus Havelund
Glenn Ammons	Arnaud Venet
Norman Cohen	Asuman Suenbuel
Thomas Mikalsen	S. Masoud Sadjadi
Paul Grace	Eric Kasten
Ant��nio Casimiro	Zhinan Zhou
Hugo Miranda	Farshad Samimi
Yuehua Lin	David Knoester
Shih-hsi Liu	Chunyang Ye
Jing Zhang	Chang Xu
Marc Schiely	Thomas Mikalsen
Jaksa Vuckovic	Norman Cohen
Partha Pal	Glenn Ammons
Paul Rubel	Chi-yin Chow
Franklin Webber	Kei Shiu Ho
Jianming Ye	Hans P. Reiser
John Zinky	

Table of Contents – Part I

OTM 2005 Keynotes

Probabilistic Ontologies and Relational Databases <i>Octavian Udrea, Deng Yu, Edward Hung, V.S. Subrahmanian</i>	1
Intelligent Web Service - From Web Services to .Plug&Play. Service Integration <i>Erich Neuhold, Thomas Risse, Andreas Wombacher, Claudia Niederée, Bendick Mahleko</i>	18
Process Modeling in Web Applications <i>Stefano Ceri</i>	20

Cooperative Information Systems (CoopIS) 2005 International Conference

CoopIS 2005 PC Co-chairs' Message	21
---	----

Workflow

Let's Go All the Way: From Requirements Via Colored Workflow Nets to a BPEL Implementation of a New Bank System <i>W.M.P. van der Aalst, J.B. Jørgensen, K.B. Lassen</i>	22
A Service-Oriented Workflow Language for Robust Interacting Applications <i>Surya Nepal, Alan Fekete, Paul Greenfield, Julian Jang, Dean Kuo, Tony Shi</i>	40
Balancing Flexibility and Security in Adaptive Process Management Systems <i>Barbara Weber, Manfred Reichert, Werner Wild, Stefanie Rinderle</i>	59

Workflow and Business Processes

Enabling Business Process Interoperability Using Contract Workflow Models <i>Jelena Zdravkovic, Vandana Kabilan</i>	77
--	----

Resource-Centric Worklist Visualisation <i>Ross Brown, Hye-young Paik</i>	94
<i>CoopFlow: A Framework for Inter-organizational Workflow Cooperation</i> <i>Issam Chebbi, Samir Tata</i>	112

Mining and Filtering

Process Mining and Verification of Properties: An Approach Based on Temporal Logic <i>W.M.P. van der Aalst, H.T. de Beer, B.F. van Dongen</i>	130
A Detailed Investigation of Memory Requirements for Publish/Subscribe Filtering Algorithms <i>Sven Bittner, Annika Hinze</i>	148
Mapping Discovery for XML Data Integration <i>Zoubida Kedad, Xiaohui Xue</i>	166

Petri Nets and Processs Management

Colored Petri Nets to Verify Extended Event-Driven Process Chains <i>Kees van Hee, Olivia Oanea, Natalia Sidorova</i>	183
Web Process Dynamic Stepped Extension: Pi-Calculus-Based Model and Inference Experiments <i>Li Zhang, Zhiwei Yu</i>	202
Petri Net + Nested Relational Calculus = Dataflow <i>Jan Hidders, Natalia Kwasnikowska, Jacek Sroka, Jerzy Tyszkiewicz, Jan Van den Bussche</i>	220

Information Access and Integrity

On the Controlled Evolution of Access Rules in Cooperative Information Systems <i>Stefanie Rinderle, Manfred Reichert</i>	238
Towards a Tolerance-Based Technique for Cooperative Answering of Fuzzy Queries Against Regular Databases <i>Patrick Bosc, Allel Hadjali, Olivier Pivert</i>	256
Filter Merging for Efficient Information Dissemination <i>Sasu Tarkoma, Jaakkko Kangasharju</i>	274

Heterogeneity

Don't Mind Your Vocabulary: Data Sharing Across Heterogeneous Peers <i>Md. Mehedi Masud, Iluju Kiringa, Anastasios Kementsietsidis</i>	292
On the Usage of Global Document Occurrences in Peer-to-Peer Information Systems <i>Odysseas Papapetrou, Sebastian Michel, Matthias Bender, Gerhard Weikum</i>	310
An Approach for Clustering Semantically Heterogeneous XML Schemas <i>Pasquale De Meo, Giovanni Quattrone, Giorgio Terracina, Domenico Ursino</i>	329

Semantics

Semantic Schema Matching <i>Fausto Giunchiglia, Pavel Shvaiko, Mikalai Yatskevich</i>	347
Unified Semantics for Event Correlation over Time and Space in Hybrid Network Environments <i>Eiko Yoneki, Jean Bacon</i>	366
Semantic-Based Matching and Personalization in FWEB, a Publish/Subscribe-Based Web Infrastructure <i>Simon Courtenage, Steven Williams</i>	385

Querying and Content Delivery

A Cooperative Model for Wide Area Content Delivery Applications <i>Rami Rashkovits, Avigdor Gal</i>	402
A Data Stream Publish/Subscribe Architecture with Self-adapting Queries <i>Alasdair J.G. Gray, Werner Nutt</i>	420
Containment of Conjunctive Queries with Arithmetic Expressions <i>Ali Kiani, Nematollaah Shiri</i>	439

Web Services, Agents

Multiagent Negotiation for Fair and Unbiased Resource Allocation <i>Karthik Iyer, Michael Huhns</i>	453
--	-----

QoS-Based Service Selection and Ranking with Trust and Reputation Management

- Le-Hung Vu, Manfred Hauswirth,
Karl Aberer* 466

An Integrated Alerting Service for Open Digital Libraries: Design and Implementation

- Annika Hinze, Andrea Schweer,
George Buchanan* 484

Security, Integrity and Consistency

Workflow Data Guards

- Johann Eder, Marek Lehmann* 502

Consistency Between e^3 -value Models and Activity Diagrams in a Multi-perspective Development Method

- Zlatko Zlatev, Andreas Wombacher* 520

Maintaining Global Integrity in Federated Relational Databases Using Interactive Component Systems

- Christopher Popfinger, Stefan Conrad* 539

Chain and Collaboration Mangement

RFID Data Management and RFID Information Value Chain Support with RFID Middleware Platform Implementation

- Taesu Cheong, Youngil Kim* 557

A Collaborative Table Editing Technique Based on Transparent Adaptation

- Steven Xia, David Sun, Chengzheng Sun,
David Chen* 576

Inter-enterprise Collaboration Management in Dynamic Business Networks

- Lea Kutvonen, Janne Metso, Toni Ruokolainen* 593

**Distributed Objects and Applications
(DOA)2005 International Conference**

- DOA 2005 PC Co-chairs' Message** 612

Web Services and Service-Oriented Architectures

Developing a Web Service for Distributed Persistent Objects in the Context of an XML Database Programming Language

Henrike Schuhart, Dominik Pietzsch,

Volker Linnemann 613

Comparing Service-Oriented and Distributed Object Architectures

Seán Baker, Simon Dobson

631

QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms

Michael C. Jaeger, Gero Mühl, Sebastian Golze

646

Multicast and Fault Tolerance

Extending the UMIOP Specification for Reliable Multicast in CORBA

Alysson Neves Bessani, Joni da Silva Fraga,

Lau Cheuk Lung 662

Integrating the ROMIOP and ETF Specifications for Atomic Multicast in CORBA

Daniel Borusch, Lau Cheuk Lung, Alysson Neves Bessani,

Joni da Silva Fraga 680

The Design of Real-Time Fault Detectors

Serge Midonnet 698

Communication Services (Was Messaging and Publish/Subscribe)

A CORBA Bidirectional-Event Service for Video and Multimedia Applications

Felipe Garcia-Sanchez, Antonio-Javier Garcia-Sanchez,

P. Pavon-Mariño, J. García-Haro 715

GREEN: A Configurable and Re-configurable Publish-Subscribe Middleware for Pervasive Computing

Thirunavukkarasu Sivaharan, Gordon Blair,

Geoff Coulson 732

Transparency and Asynchronous Method Invocation

Pierre Vignéras 750

Techniques for Application Hosting

- COROB: A Controlled Resource Borrowing Framework for Overload Handling in Cluster-Based Service Hosting Center
Yufeng Wang, Huaimin Wang, Dianxi Shi, Bixin Liu 763

- Accessing X Applications over the World-Wide Web
Arno Puder, Siddharth Desai 780

- Exploiting Application Workload Characteristics to Accurately Estimate Replica Server Response Time
Corina Ferdean, Mesaac Makpangou 796

Mobility

- Automatic Introduction of Mobility for Standard-Based Frameworks
Grègory Haïk, Jean-Pierre Briot, Christian Queinnec 813

- Empirical Evaluation of Dynamic Local Adaptation for Distributed Mobile Applications
Pablo Rossi, Caspar Ryan 828

- Middleware for Distributed Context-Aware Systems
Karen Henricksen, Jadwiga Indulksa, Ted McFadden, Sasitharan Balasubramaniam 846

- Timely Provisioning of Mobile Services in Critical Pervasive Environments
Filippos Papadopoulos, Apostolos Zarras, Evaggelia Pitoura, Panos Vassiliadis 864

- Mobility Management and Communication Support for Nomadic Applications
Marcello Cinque, Domenico Cotroneo, Stefano Russo 882

- Platform-Independent Object Migration in CORBA
Rüdiger Kapitza, Holger Schmidt, Franz J. Hauck 900

- Author Index** 919

Table of Contents – Part II

Distributed Objects and Applications (DOA) 2005 International Conference (continued)

Security and Data Persistence

- Evaluation of Three Approaches for CORBA Firewall/NAT Traversal
Antonio Theophilo Costa, Markus Endler, Renato Cerqueira 923

- On the Design of Access Control to Prevent Sensitive Information Leakage in Distributed Object Systems: A Colored Petri Net Based Model
Panagiotis Katsaros 941

- Offline Business Objects: Enabling Data Persistence for Distributed Desktop Applications
Pawel Gruszczynski, Stanislaw Osinski, Andrzej Swedrzynski 960

Component Middleware

- Middleware Support for Dynamic Component Updating
Jaiganesh Balasubramanian, Balachandran Natarajan, Douglas C. Schmidt, Aniruddha Gokhale, Jeff Parsons, Gan Deng 978

- Two Ways of Implementing Software Connections Among Distributed Components
Selma Matougui, Antoine Beugnard 997

- On the Notion of Coupling in Communication Middleware
Lachlan Aldred, Wil M.P. van der Aalst, Marlon Dumas, Arthur H.M. ter Hofstede 1015

Java Environments

- A Task-Type Aware Transaction Scheduling Algorithm in J2EE
Xiaoning Ding, Xin Zhang, Beihong Jin, Tao Huang 1034

- Application Object Isolation in Cross-Platform Operating Environments
Stefan Paal, Reiner Kammüller, Bernd Freisleben 1046

Garbage Collection in the Presence of Remote Objects: An Empirical Study <i>Witawas Srisa-an, Mulyadi Oey, Sebastian Elbaum</i>	1065
Peer-to-Peer Computing Architectures	
Peer-to-Peer Distribution Architectures Providing Uniform Download Rates <i>Marc Schiely, Pascal Felber</i>	1083
JXTA Messaging: Analysis of Feature-Performance Tradeoffs and Implications for System Design <i>Emir Halepovic, Ralph Deters, Bernard Traversat</i>	1097
Aspect Oriented Middleware	
An Aspect-Oriented Communication Middleware System <i>Marco Tulio de Oliveira Valente, Fabio Tirelo, Diana Campos Leao, Rodrigo Palhares Silva</i>	1115
Using AOP to Customize a Reflective Middleware <i>Nélio Cacho, Thaís Batista</i>	1133
Ontologies, Databases and Applications of Semantics (ODBASE) 2005 International Conference	
ODBASE 2005 PC Co-Chairs' Message	1151
Information Integration and Modeling	
Inferring Complex Semantic Mappings Between Relational Tables and Ontologies from Simple Correspondences <i>Yuan An, Alex Borgida, John Mylopoulos</i>	1152
Ontology Transformation and Reasoning for Model-Driven Architecture <i>Claus Pahl</i>	1170
Multidimensional RDF <i>Manolis Gergatsoulis, Pantelis Lilis</i>	1188
GeRoMe: A Generic Role Based Metamodel for Model Management <i>David Kensche, Christoph Quix, Mohamed Amine Chatti, Matthias Jarke</i>	1206

Query Processing

Probabilistic Iterative Duplicate Detection <i>Patrick Lehti, Peter Fankhauser</i>	1225
Efficient Processing of XPath Queries with Structured Overlay Networks <i>Gleb Skobeltsyn, Manfred Hauswirth, Karl Aberer</i>	1243
Community Based Ranking in Peer-to-Peer Networks <i>Christoph Tempich, Alexander Löser, Jörg Heizmann</i>	1261
Ontology-Based Representation and Query of Colour Descriptions from Botanical Documents <i>Shenghui Wang, Jeff Z. Pan</i>	1279

Ontology Construction

Creating Ontologies for Content Representation—The OntoSeed Suite <i>Elena Paslaru Bontas, David Schlangen, Thomas Schrader</i>	1296
Fully Automatic Construction of Enterprise Ontologies Using Design Patterns: Initial Method and First Experiences <i>Eva Blomqvist</i>	1314
Automatic Ontology Extraction from Unstructured Texts <i>Khurshid Ahmad, Lee Gillam</i>	1330

Metadata

Metadata Management in a Multiversion Data Warehouse <i>Robert Wrembel, Bartosz Bębel</i>	1347
Metadata Management for Ad-Hoc InfoWare - A Rescue and Emergency Use Case for Mobile Ad-Hoc Scenarios <i>Norun Sanderson, Vera Goebel, Ellen Munthe-Kaas</i>	1365
Managing Petri Nets in MOF Repositories <i>Hélio L. dos Santos, Paulo R.M. Maciel, Nelson S. Rosa, Roberto S.M. Barros</i>	1381
A Meta-ontological Architecture for Foundational Ontologies <i>Heinrich Herre, Frank Loewe</i>	1398

Information Retrieval and Classification

Web Image Semantic Clustering <i>Zhiguo Gong, Leong Hou U, Chan Wa Cheang</i>	1416
Biomedical Retrieval: How Can a Thesaurus Help? <i>Leonie IJzereef, Jaap Kamps, Maarten de Rijke</i>	1432
Hybrid Model for Semantic Similarity Measurement <i>Angela Schwering</i>	1449
Ontology-Based Spatial Query Expansion in Information Retrieval <i>Gaihua Fu, Christopher B. Jones, Alia I. Abdelmoty</i>	1466
Security Ontology for Annotating Resources <i>Anya Kim, Jim Luo, Myong Kang</i>	1483

System Verification and Evaluation

An Ontology for Mobile Agents in the Context of Formal Verification <i>Paulo Salem da Silva, Ana Cristina Vieira de Melo</i>	1500
Evaluating Ontology Criteria for Requirements in a Geographic Travel Domain <i>Jonathan Yu, James A. Thom, Audrey Tam</i>	1517
A Self-monitoring System to Satisfy Data Quality Requirements <i>Cinzia Cappiello, Chiara Francalanci, Barbara Pernici</i>	1535

Active Rules and Web Services

An Ontology- and Resources-Based Approach to Evolution and Reactivity in the Semantic Web <i>Wolfgang May, José Júlio Alferes, Ricardo Amador</i>	1553
Automatic Web Service Composition Based on Graph Network Analysis Metrics <i>John Gekas, Maria Fasli</i>	1571

ODBASE 2005 Short Papers

Two Reasoning Methods for Extended Fuzzy ALCH <i>Dazhou Kang, Jianjiang Lu, Baowen Xu, Yanhui Li, Yanxiang He</i>	1588
Expressing Preferences in a Viewpoint Ontology <i>Rallou Thomopoulos</i>	1596
Architecting Ontology for Scalability and Versatility <i>Gang Zhao, Robert Meersman</i>	1605
OWL-Based User Preference and Behavior Routine Ontology for Ubiquitous System <i>Kim Anh Pham Ngoc, Young-Koo Lee, Sung-Young Lee</i>	1615
Reasoning on Dynamically Built Reasoning Space with Ontology Modules <i>Fabio Porto</i>	1623
An Efficient Branch Query Rewriting Algorithm for XML Query Optimization <i>Hyoseop Shin, Minsoo Lee</i>	1629
Automated Migration of Data-Intensive Web Pages into Ontology-Based Semantic Web: A Reverse Engineering Approach <i>Sidi Mohamed Benslimane, Mimoun Malki, Djamel Amar Bensaber</i>	1640
Author Index	1651

Probabilistic Ontologies and Relational Databases

Octavian Udrea¹, Deng Yu¹, Edward Hung², and V.S. Subrahmanian¹

¹ University of Maryland, College Park MD 20742, USA
`{udrea, yuzi, vs}@cs.umd.edu`

² The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong^{*}
`csehung@comp.polyu.edu.hk`

Abstract. The relational algebra and calculus do not take the semantics of terms into account when answering queries. As a consequence, not all tuples that should be returned in response to a query are always returned, leading to low recall. In this paper, we propose the novel notion of a *constrained probabilistic ontology* (CPO). We developed the concept of a CPO-enhanced relation in which each attribute of a relation has an associated CPO. These CPOs describe relationships between terms occurring in the domain of that attribute. We show that the relational algebra can be extended to handle CPO-enhanced relations. This allows queries to yield sets of tuples, each of which has a probability of being correct.

1 Introduction

Over the years, there has been an implicit assumption that relational databases correctly answer queries. However, relational databases are being increasingly populated by people who know very little about how databases work. They are also being used more and more by people who do not know much about the innards of a DBMS. It is therefore important that the DBMS must be smart enough to return answers that have *both* a high precision and a high recall. Databases today do very well in terms of precision, but not as well with respect to recall. Our goal in this paper is to *compute* probabilistic answers to queries internally, but only return a tuple to the user if its probability of being correct exceeds a given threshold p_{thr} . As a consequence, the end user does not even see the probabilities (either in the data or in the answer) unless he really wants to. For example, suppose the user poses a query Q . Our PARQ (Probabilistic Answers to Relational Queries) system may compute that tuple t should be in the answer with 90% probability. If the threshold $p_{thr} \leq 0.9$, then t will be returned to the user, otherwise not. Thus, PARQ's use of probabilities is completely hidden from the user - the inputs are the same old relations and queries, and the output is also just an ordinary relation. The probabilities are only used internally by the system. We should emphasize, however, that a DB administrator can and should have visibility into some of the workings of PARQ.

* Work performed while at the University of Maryland, College Park MD 20742, USA.

This raises two questions: (I) How should we assign a probability to a tuple t being in the answer to query Q , and (II) how should the threshold p_{thr} be set? Most of this paper will focus on answering question (I). Question (II) will be answered experimentally towards the end of this paper.

The organization and contributions of this paper are summarized below.

1. We first provide a detailed motivating example from the domain of astronomy in Section 2.
2. Our first major contribution is the concept of a *constrained probabilistic ontology* (CPO) introduced in Section 3. We introduce the notion of a *CPO-enhanced relation* in which a CPO is associated with each attribute of a relation.
3. In Section 4, we provide a formal definition of what it means for a CPO to be consistent. We show that checking consistency of CPOs is NP-complete. Fortunately, there is a class of CPOs that are guaranteed to be consistent and there are polynomial algorithms to check if a given CPO belongs to this class or not.
4. When performing various operations, we might need to merge two CPOs together. For example, when doing a condition-join between two CPO-enhanced relations with the condition $A_1 = A_2$, we might need to merge the CPOs associated with A_1 and A_2 . We develop the **CPOMerge** algorithm that merges two CPOs together in the presence of certain “interoperation constraints” linking together terms in the two ontologies in Section 5.
5. In Section 6, we extend the relational algebra to utilize CPOs. In particular, we define selection, join, and the set operations. Projection is unaffected by CPOs.

2 Motivating Example

There are numerous astronomical databases¹ in existence today. These, in turn, are accompanied by several *stellar classification schemes* based on diverse characteristics such as luminosity, mass, size, composition, etc. For instance, the most popular scientific classifiers are the *spectral class classification* and the *Yerkes luminosity classes*. Uncertainty in star classifications occurs for many reasons: first, some celestial bodies are discovered purely by mathematical means and most classifiers are based on direct observation of the body in question. Secondly, even with the possibility of direct observation there is often not enough information to assume that a star belongs to some specialized class. One may also need to classify a star according to multiple scientific classifiers at once. Figure 1 (without the underlined elements) shows a simple CPO (we will give the formal details later) for holding star data, together with an ontology for star classification. The stars are classified according to size, brightness and color. Each such *decomposition* is *disjoint* and mutually disjoint classes are joined by a “d”. For instance, stars that are **red** cannot be **yellow** at the same time. We can easily see that the ontology in this example is an **is-a** graph with arrows indicating the relationship. The labels on the edges of the graph have an intuitive

¹ A quick Google search on *astronomy database* reveals over a million hits - the first five pages returned led to several databases.

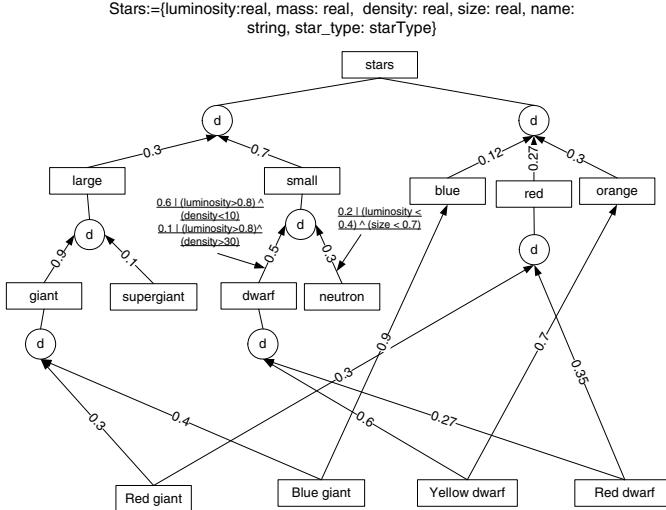


Fig. 1. A simple CPO

meaning: if edge $u \rightarrow v$ is labeled with quantity p then there is a p probability that an arbitrary object in class v is in fact in class u . The underlined elements in Figure 1 represent *constrained probabilities*, which give circumstances in which the default probability of an edge may change. Using such an ontology, users may ask queries such as:

- **Select all stars that are small.** Clearly, any entity listed explicitly as a small star should be returned. However, in addition, any entity listed as a **dwarf**, **neutron**, etc. should be returned using class-subclass relationships. Now consider an entity e listed in a database as a **star**. There is a 70% probability that any star is small - so if $p_{thr} \leq 0.7$, e should be returned as a response to this query.
- **Join example.** Given two relations containing star data with attributes $\mathcal{R}_1 = \{\text{name}, \text{density}, \text{star_type}\}$ and $\mathcal{R}_2 = \{\text{mass}, \text{location}, \text{star_type}\}$, we want to join these relations based on the **star_type** attribute. Classical relational join cannot capture the fact that different values in the **star_type** attribute can in fact refer to the same set of objects and thus will not return all tuples. If we know for instance that a **red** star is the same as a **crimson** star, using CPO enhanced relations we can also join the tuples that have $\mathcal{R}_1.\text{star_type} = \text{red}$ and $\mathcal{R}_2.\text{star_type} = \text{crimson}$.
- **Select all stars that are dwarf with a 50% probability or more.** This kind of query cannot be answered using a relation alone (with no probabilities encoded in it). For a simple condition such as $\text{star_type} = \text{dwarf}$ it adds a *certainty threshold*. The result of such an operation would consist of all stars that have **star_type** either **dwarf**, **yellow_dwarf**, **red_dwarf** as well as all

stars that have `star_type=small`, since there is a 0.5 probability that these are `dwarf` as well. We define such a selection operation in Section 6.

We will use these examples throughout the rest of this article to illustrate our definitions. By the end of the article we will have defined selection, projection, join and set operations for CPO-enhanced relations.

3 Constrained Probabilistic Ontology (CPO)

Eiter et. al. [1] proposed the concept of a *probabilistic object base* (POB) in which they defined a structure called a POB-schema. They then showed how to embed probabilities into object bases. In contrast, we will slightly modify their POB-schema definition and then use it to define our main structure, a CPO. Of course, we use CPOs in a very different way than Eiter et. al.[1] — we use it to improve the recall of answers to queries posed to relational databases with no probabilities anywhere in the DB.

Definition 1. (*POB-schema*). A *POB-schema* is a quadruple $(\mathcal{C}, \Rightarrow, \text{me}, \psi)$ where:

- (1) \mathcal{C} is a finite set of classes.
- (2) \Rightarrow is a binary relation on \mathcal{C} such that $(\mathcal{C}, \Rightarrow)$ is a directed acyclic graph. The relation $c_1 \Rightarrow c_2$ says that the class c_1 is an immediate subclass of c_2 .
- (3) me maps each class c to a set of disjoint subsets of the immediate subclasses of c . In our example, $\text{me}(\text{stars}) = \{\{\text{large}, \text{small}\}, \{\text{blue}, \text{red}, \text{orange}\}\}$. me produces clusters corresponding to the disjoint decompositions of a class.
- (4) ψ maps each edge in $(\mathcal{C}, \Rightarrow)$ to a positive rational number in the interval $[0,1]$ such that for all classes c and all clusters $\mathcal{L} \in \text{me}(c)$, it is the case that $\sum_{d \in \mathcal{L}} \psi(d, c) \leq 1$. This property will ensure that the sum of the probabilities on the edges originating from a disjoint decomposition is less than or equal to 1.

If we strip away all the conditional probabilities in Figure 1 (i.e. the condition probabilities from the node `small` to `dwarf` and `neutron`, then we would have a POB-schema. Our definition is identical to that in [1] except for the tuple types associated with the classes in the graph. We now define the transitive closure of the \Rightarrow relation in a POB-schema.

Definition 2. (\Rightarrow^*). The \Rightarrow^* relation is the reflexive, transitive closure of \Rightarrow .

A CPO is obtained from a POB-schema by allowing edges to be labeled with multiple *constrained/conditional probabilities*. When associating a relation with a POB-schema, one may want to express that when certain conditions are met, the probability of an object being in a certain subclass of a given class increases or decreases. For instance, in Figure 1, we might want to express the fact that when $\text{luminosity} > 0.8$ (which is true of our sun), the probability that a `small` star is a `dwarf` increases to 0.75.

Definition 3. (*Simple constraint*). A simple constraint is of the form $\mathcal{A}_i \in \mathcal{D}$, where \mathcal{A}_i is an attribute name, and $\mathcal{D} \subset \text{dom}(\mathcal{A}_i)$. A particular case of a simple constraint is the *nil constraint*, defined by $\mathcal{A}_i \in \text{dom}(\mathcal{A}_i)$. For a set of objects \mathcal{O} , we denote by \mathcal{O}_γ the subset of objects for which constraint γ holds. Two constraints γ_1, γ_2 are *disjoint* if for all sets of objects \mathcal{O} , $\mathcal{O}_{\gamma_1} \cap \mathcal{O}_{\gamma_2} = \emptyset$.

The definition above is a generalization of the normal comparison operators, in that every comparison of an attribute to a value can be re-written in the form of a simple constraint. For the rest of the paper, we will abuse notation and use the usual comparison operators to denote simple constraints.

Definition 4. (*Constraint probability*). A constraint probability is a pair (p, γ) , where p is a rational number in the interval $[0, 1]$ and γ is a conjunction of simple constraints. (p, γ) is called a *nil constraint probability* when γ is true. We will denote these probabilities by (p, true) .

Example 1. For example, in Figure 1, $(0.6, \text{luminosity} > 0.8)$ is a constraint probability. The 0.3 (or $(0.3, \text{true})$) label between nodes `orange` and `stars` is a nil constraint probability. Intuitively, the constraint probability $(0.6, \text{luminosity} > 0.8)$ for the edge `dwarf` \Rightarrow `small` states that for objects classified as `small` that have *luminosity* > 0.8 , there is a 0.6 probability that these are in fact `dwarf`. Thus, additional information allows us to refine the default probability of 0.5 for the respective edge. We will now discuss the concept of *labeling* for the edges of a CPO. In the following definition, we denote by $E(G)$ the set of edges for a graph G .

Definition 5. (*Labeling*). Suppose G is a graph whose set of edges is $E(G)$, and suppose Γ is the set of all possible constraint probabilities associated with some set of attributes. A labeling of G is any mapping $\varphi : E(G) \rightarrow 2^\Gamma$. As usual, 2^Γ is the powerset of Γ .

Definition 6. (*Valid labeling*). A labeling function φ with respect to a graph G , a relation \mathcal{R} and an attribute $\mathcal{A} \in \mathcal{R}$ is called a **valid labeling** if:

- (V1) \forall edges $(c, d) \in E(G), \forall (p, \gamma) \in \varphi(c, d), \gamma$ refers only to attributes in $\mathcal{R} - \{\mathcal{A}\}$;
- (V2) \forall edges $(c, d) \in E(G)$, there is exactly one nil constraint probability in $\varphi(c, d)$.

A valid labeling maps each edge of a graph to a set of constraint probabilities. The set of constraint probabilities for an edge $c \Rightarrow d$ contains only one nil constraint probability – as this represents the probability of an object in d being in c , it must be unique. Intuitively, \mathcal{A} will be the attribute whose domain of values is represented by the ontology, and thus we require that any simple constraints do not refer to the value of \mathcal{A} itself.

We are now ready to define a CPO based on the concepts of constraint probabilities and labeling. A CPO can be informally defined as a POB-schema with edges labeled by multiple constraint probabilities. The valid labeling property will ensure that we have at least one nil constraint probability on every edge. Thus, a CPO is a generalization of a POB-schema.

Definition 7. (*Constraint Probabilistic Ontology*). A CPO with respect to a relation \mathcal{R} and an attribute $\mathcal{A} \in \mathcal{R}$ is a quadruple $(\mathcal{C}, \Rightarrow, \text{me}, \wp)$ where:

- (1) \mathcal{C} is a finite set of classes. \mathcal{C} is a superset of the domain of \mathcal{A} . We should note that a CPO can be defined in such manner as to be associated with more than one attribute of a relation. This is in fact the case with our implementation. However, for the purpose of simplicity the discussion will assume that a CPO is associated with only one attribute within a relation.
- (2) \Rightarrow is a binary relation on \mathcal{C} such that $(\mathcal{C}, \Rightarrow)$ is a directed acyclic graph. The relation $c_1 \Rightarrow c_2$ says that the class c_1 is an immediate subclass of c_2 .
- (3) me maps each class c to a partition of the set of all immediate subclasses of c .
- (4) \wp is a valid labeling with respect to the graph $(\mathcal{C}, \Rightarrow)$, relation \mathcal{R} and attribute \mathcal{A} .

Definition 8. (*CPO enhanced relation*). A CPO enhanced relation is a triple (T, \mathcal{R}, f) , where T is a set of CPOs, \mathcal{R} is a relation with schema $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ and $f : \{\mathcal{A}_1, \dots, \mathcal{A}_n\} \rightarrow T \cup \{\perp\}$ is a mapping called the mapping function such that:

- $\forall S \in T, \exists j$ such that $f(\mathcal{A}_j) = S$. Intuitively, this means that any CPO in T has at least one attribute it is associated with.
- $\forall S \in T$, suppose $f(\mathcal{A}_i) = S$. We then require that $\text{dom}(\mathcal{A}_i) \subseteq \mathcal{C}_S$. Intuitively, this means that any CPO is well defined with respect to the relation and all attributes it is associated with.

Example 2. Suppose S denotes the CPO in Figure 1. Then $(\{S\}, \text{Stars}, f)$ is a CPO-enhanced relation, with $f(\text{star_type}) = S$ and for any other attribute $\mathcal{A} \in \text{Stars}$, $\mathcal{A} \neq \text{star_type}$, $f(\mathcal{A}) = \perp$. The conditions above are satisfied as there is an attribute that maps to S and S is well defined with respect to *Stars* and the *star_type* attribute.

Definition 9. (CPO^k). A k -order CPO (or CPO^k for short) is a CPO such that for all edges (c, d) in the CPO, $|\wp(c, d)| \leq k$. In particular, a CPO^1 is a simple POB.

Intuitively, a k -order CPO is one that associates at most k constraint probabilities with each edge.

Definition 10. (*Probability path*). For a CPO $S = (\mathcal{C}, \Rightarrow, \text{me}, \wp)$, we say that two classes $c, d \in \mathcal{C}$ have a path of probability q (written $c \rightsquigarrow_q d$) if $c \Rightarrow^* d$ through c_1, \dots, c_k and there is a function g defined on the set of edges in the c, c_1, \dots, c_k, d chain, such that $g(x, y) \in \wp(x, y) \forall (x, y)$ edges along the chain (i.e. g selects one constraint probability from each edge along the path) and $\prod_{g(x, y) = (p, \gamma)} p \geq q$. We also denote by $\Gamma_g(c \rightsquigarrow_q d)$ the set of constraints selected by g along the path.

We will often abuse notation and just use (T, \mathcal{R}) to denote a CPO-enhanced relation, leaving f implicit.

4 Consistency

Intuitively, a CPO is consistent if for any universe of objects \mathcal{O} , we can find an assignment of objects to the classes of the CPO that satisfy the conditions required by subclassing and constraint probabilities. As proven in [1], consistency checking for POBs is NP-complete, thus we expect a similar result for CPOs.

Definition 11. (*Consistent CPO*). Let $\mathbf{S} = (\mathcal{C}, \Rightarrow, me, \wp)$ be a CPO. An interpretation of \mathbf{S} is a mapping ϵ from \mathcal{C} to the set of all finite subsets of a set \mathcal{O} (an arbitrary universe of objects). An interpretation ϵ of \mathbf{S} is called a taxonomic probabilistic model if and only if:

- (C1) $\epsilon(c) \neq \emptyset, \forall c \in \mathcal{C}$;
- (C2) $\epsilon(c) \subseteq \epsilon(d), \forall c, d \in \mathcal{C}, c \Rightarrow d$
- (C3) $\epsilon(c) \cap \epsilon(d) = \emptyset, \forall c, d \in \mathcal{C}, c \neq d$ that belong to the same cluster $\mathcal{L} \in \cup me(\mathcal{C})$.
- (C4) \forall edges $u \Rightarrow v \in (\mathcal{C}, \Rightarrow)$, \forall constraint probabilities $(p, \gamma) \in \wp(u, v)$, $|\epsilon(u)| \geq p * |\epsilon(v)|_\gamma$, where by $|\epsilon(v)|_\gamma$ we denote the number of objects assigned to class v that meet constraint γ . All objects in a class meet the nil constraint.

A CPO is consistent if and only if it has a taxonomic probabilistic model.

Theorem 1. The problem of deciding whether a given CPO \mathbf{S} is consistent is NP-complete.

The problem is NP-complete since the NP-complete problem of deciding whether a weight formula is satisfiable in a measurable probability structure [2] can be polynomially reduced to consistency checking for a CPO. Furthermore, a non-deterministic algorithm that performs consistency checking in polynomial time can be devised.

Nonetheless, polynomial time algorithms for deciding the consistency of a CPO in relevant special cases may be possible. Eiter et al. [1, 3] introduced the concept of a pseudoconsistent POB — we extend their definition to the case of a CPO and then present a notion of well structured CPOs. A pseudoconsistent and well structured CPO is guaranteed to be consistent. We will introduce the concept of a pseudoconsistent CPO that encompasses most of the properties for pseudoconsistent and well structured POB schemas. Furthermore, we will define our own concept of well-structuredness.

Definition 12. (*Pseudoconsistent CPO*). For a CPO $\mathbf{S} = (\mathcal{C}, \Rightarrow, me, \wp)$, we denote $\mathbf{S}^*(c) = \{d \in \mathcal{C} | d \Rightarrow^* c\}$. \mathbf{S} is pseudoconsistent if (rule labeling follows the one in [1]):

- (P2) For all classes $c \in \mathcal{C}$ and for all clusters $\mathcal{L} \in me(c)$, no two distinct classes $c_1, c_2 \in \mathcal{L}$ have a common subclass.
- (W1) The $(\mathcal{C}, \Rightarrow)$ graph has a top (or root) element.
- (W2) For every class $c \in \mathcal{C}$, any two distinct immediate subclasses have either no common subclass or a greatest common subclass which is different from them.

- (W3) For every class $c \in \mathcal{C}$, the undirected graph $G_S(c) = (\nu, \epsilon)$ that is defined by $\nu = \text{me}(c)$ and $\epsilon = \{\{\mathcal{L}_1 \times \mathcal{L}_2\} \in \nu \times \nu \mid \mathcal{L}_1 \neq \mathcal{L}_2, \bigcup S^*(\mathcal{L}_1) \cap \bigcup S^*(\mathcal{L}_2) \neq \emptyset\}$ is acyclic (i.e., for every class $c \in \mathcal{C}$, the partition clusters in $\text{me}(c)$ are not cyclically connected through common subclasses). In short, multiple inheritance does not cyclically connect partition clusters.
- (W4) $\forall c \in \mathcal{C}$, if the graph $G_S(c)$ has an edge (i.e. two distinct clusters in $\text{me}(c)$ have a common subclass), then every path from a subclass of c to the top element of $(\mathcal{C}, \Rightarrow)$ goes through c . In short, multiple inheritance can be locally isolated in the graph $(\mathcal{C}, \Rightarrow)$.

Finding a method that checks for consistency in polynomial time requires that we “summarize” the edge labeling information. Even though we lose information through this process, we can determine a subset of CPOs that allow for a reasonable consistency checking algorithm.

Definition 13. (Weight factor). Let P be a set of non-nil constraint probabilities. Then we define a weight factor $w_f(P)$ as follows:

- If $P = \emptyset$, then $w_f(P) = 0$;
- If $P = \{(p, \gamma)\}$, then $w_f(P) = p$;
- Otherwise, we define w_f recursively with the formula: $w_f(P \cup Q) = w_f(P) + w_f(Q) - w_f(P) \cdot w_f(Q)$.

Lema 1 Let \mathcal{O} be an arbitrary set of objects and let (p_1, γ_1) , (p_2, γ_2) be two non-nil constraint probabilities. Then we can always select a set \mathcal{O}' with $|\mathcal{O}'| \leq (p_1 + p_2 - p_1 \cdot p_2) \cdot |\mathcal{O}|$ such that:

- \mathcal{O}' contains at least $p_1 \cdot |\mathcal{O}_{\gamma_1}|$ objects that meet constraint γ_1 (\mathcal{O}_{γ_1} is the set of objects from \mathcal{O} that meet γ_1);
- \mathcal{O}' contains at least $p_2 \cdot |\mathcal{O}_{\gamma_2}|$ objects that meet constraint γ_2 ;

Theorem 2. Given a **consistent** CPO $S = (\mathcal{C}, \Rightarrow, \text{me}, \wp)$ and $c \Rightarrow d$ an edge in $(\mathcal{C}, \Rightarrow)$, we write $\wp(c, d) = \{(p_0, \text{true})\} \cup \wp'(c, d)$ (i.e. we isolate the nil constraint probability). Then condition (C4) of Definition 11 for $c \Rightarrow d$ can be satisfied by assigning $w(c, d) = \max(p_0, w_f(\wp'(c, d)))$ objects from class d into class c . (Note: this condition can be satisfied by selecting fewer objects in some instances, but as mentioned before we are trying to identify a subset of CPO viable for consistency checking).

The proof is based on induction on the size of $\wp'(c, d)$.

Definition 14. (Well structured CPO). A CPO $S = (\mathcal{C}, \Rightarrow, \text{me}, \wp)$ is well-structured if the following conditions hold:

- (W5) $\forall d, c_1, c_2 \in \mathcal{C}$ such that $c_1 \Rightarrow d, c_2 \Rightarrow d, c_1 \neq c_2$ and c_1, c_2 are in the same cluster $\mathcal{L} \in \text{me}(d)$ and $\forall (p_1, \gamma_1) \in \wp(c_1, d), \forall (p_2, \gamma_2) \in \wp(c_2, d)$, if $\gamma_1, \gamma_2 \neq \text{true}$, then γ_1 and γ_2 are **disjoint**.
- (W6) $\forall d \in \mathcal{C}, \forall \mathcal{L} \in \text{me}(d), \forall c \in \mathcal{L}, \sum_c w(c, d) \leq 1$.

Theorem 3. *A well structured and pseudoconsistent CPO is consistent.*

The proof is immediate. Conditions of pseudoconsistency account for (C1)–(C3) in Definition 11, as shown in [1]. Conditions (W5) and (W6) account for (C4) in Definition 11.

We can now give an algorithm for consistency checking. Such an algorithm decides on consistency by checking that a CPO belongs to the set defined in the previous theorem. We will use only such CPOs for the discussions and experiments to follow. The *Consistency-Check(S)* algorithm is two fold: the first part determines whether \mathbf{S} is pseudoconsistent according to Definition 12. This algorithm is the *well-structured(S)* algorithm defined for POB schemas, implemented with a simple operation to determine the top (if any) of a CPO schema. We can simply reuse this algorithm since it only deals with properties involving inheritance and not with edge labeling. The *well-structured(S)* algorithm runs in $\mathcal{O}(n^2e)$ time, where $n = |\mathcal{C}|$ and e is the number of directed edges in $(\mathcal{C}, \Rightarrow)$. We will now define a new algorithm that checks for conditions (W5) and (W6). The algorithm is described below.

Algorithm: well-structured-CPO(S)

Input: Pseudoconsistent CPO $\mathbf{S} = (\mathcal{C}, \Rightarrow, \text{me}, \wp)$.

Output: true if \mathbf{S} is well-structured and false otherwise.

Notation: We denote by $\text{top}(\mathbf{S})$ the top element of $(\mathcal{C}, \Rightarrow)$. Q is a FIFO queue, initially empty.

```

1) Q ← {top(S)};
2) while ¬empty(Q) do
3)   c ← dequeue(Q);
4)   for each L ∈ me(c) do
5)     Q ← enqueue(L, Q);    (add all elements within L to the queue)
6)   for each d1, d2 ∈ L, d1 ≠ d2 do
7)     for each (p1, γ1), (p2, γ2) ∈ ϕ(d1, c) × ϕ(d2, c), γ1, γ2 ≠ nil do
8)       if γ1, γ2 are not disjoint, then
9)         return false; (S does not satisfy (W5))
10)      end;
11)      end;
12)      Sum ← 0;
13)      for each d ∈ L do
14)        Compute w(d, c);
15)        Sum ← Sum + w(d, c).
16)      end;
17)      if Sum > 1 then return false; (S does not satisfy (W6))
18)    end;
19)  end;
20) return true (S is a well structured CPO);

```

It is easy to see that the algorithm described here runs in $\mathcal{O}(n^2k^2)$ for a CPO^k . The **for** statements on lines 6 and 7 account for this limit. Computation of $w(d, c)$ is done in $\mathcal{O}(k)$ for every class d , according to the recursive definition given.

So far, we have extended the POB-schema defined by Eiter et. al. [1] in several ways. First, we have added constrained probabilities to the ontology. Second, we have generalized the well-structuredness property, by allowing different paths between two classes to have different probabilities. In the following sections, we introduce new concepts that allow us to extend the relational algebra to handle CPOs.

5 CPO Merging

Suppose we wish to perform a natural join operation on two relations, $\mathcal{R} = \mathcal{R}_1 \Theta_{\mathcal{A}} \mathcal{R}_2$. Suppose $\mathcal{R}_1.\mathcal{A}$ is associated with CPO S_1 and $\mathcal{R}_1.\mathcal{A}$ is associated with CPO S_2 . As each attribute has only one associated CPO, we need a method to integrate S_1 and S_2 . We will declaratively define what the integration of two ontologies is, and then provide an algorithm to implement it. The notion of an inter-operation constraint specifies any known relationships between terms in the two ontologies.

Definition 15. (*Interoperation constraints*). *For the purpose of merging two CPOs $S_1 = (\mathcal{C}_1, \Rightarrow_1, \text{me}_1, \wp_1)$, $S_2 = (\mathcal{C}_2, \Rightarrow_2, \text{me}_2, \wp_2)$, a set of interoperation constraints will contain elements of the following type:*

- (*Equality constraint*). $c_1 := c_2$, where $c_1 \in \mathcal{C}_1$ and $c_2 \in \mathcal{C}_2$. This type of constraint specifies that two classes from different CPOs always refer to the same set of objects. In order to avoid situations where by transitivity of the equality constraint we would have two classes from the same CPO in an equality relation, we require that each class from each CPO appear at most once in the set of equality constraints.
- (*Immediate subclassing constraint*). *Such constraints are in the form $c_1 \sqsubset_{(p_0, \text{true}), (p_1, \gamma_1), \dots, (p_k, \gamma_k)} c_2$, where $c_1 \in \mathcal{C}_1$, $c_2 \in \mathcal{C}_2$ and $(p_0, \text{true}), (p_1, \gamma_1), \dots, (p_k, \gamma_k)$ are constraint probabilities such that there is exactly one nil constraint probability.* This constraint states that c_1 should be an immediate subclass of c_2 in the merged ontologies with the given labeling on the edge between the two.

The subclassing constraints specify a relation of *immediate subclassing*. This is done firstly in order to simplify the merge algorithm. Secondly, in the extended version of the paper available at <http://www.cs.umd.edu/users/udrea/ProbOntologies.pdf> we provide algorithms to infer equality constraints and thus immediate subclassing constraints would be used when provided by a DB administrator.

Definition 16. (*Integration witness*). *Given two CPOs, $S_1 = (\mathcal{C}_1, \Rightarrow_1, \text{me}_1, \wp_1)$, $S_2 = (\mathcal{C}_2, \Rightarrow_2, \text{me}_2, \wp_2)$ and a set of interoperation constraints \mathcal{I} defined on S_1, S_2 , we define a witness to the integration of S_1, S_2 as $S \doteq \mu_{\mathcal{I}}(S_1, S_2)$ if the following conditions hold:*

- (O1) *S is a CPO $(\mathcal{C}, \Rightarrow, \text{me}, \wp)$.* This states that the result of merging two CPOs is a CPO, thus abiding by Definition 7.

- (O2) $\forall c \in \mathcal{C}_1 \cup \mathcal{C}_2$, only one of the following holds: either $c \in \mathcal{C}$ or $\exists d \in \mathcal{C}_1 \cup \mathcal{C}_2$ such that $d \in \mathcal{C}$ and $(c := d) \in \mathcal{I}$. This condition states that all classes from both ontologies would be present in the result, either directly or through a representative if they are part of an equality constraint. We will denote by $CR(x)$ the representative of class x in \mathcal{C} .
- (O3) $\forall c_1 \Rightarrow_1 d_1 \in (\mathcal{C}_1, \Rightarrow_1), c_2 \Rightarrow_2 d_2 \in (\mathcal{C}_2, \Rightarrow_2), CR(c_1) \Rightarrow CR(d_1), CR(c_2) \Rightarrow CR(d_2) \in (\mathcal{C}, \Rightarrow)$. Intuitively, this states that no edges are lost during the merging process.
- (O4) $\forall c_1 \Rightarrow_1 d_1 \in (\mathcal{C}_1, \Rightarrow_1), c_2 \Rightarrow_2 d_2 \in (\mathcal{C}_2, \Rightarrow_2)$ such that $\{c_1 := c_2, d_1 := d_2\} \subseteq \mathcal{I}$, and $\forall (p_1, \gamma_1) \in \wp_1(c_1, d_1)$ and $(p_2, \gamma_2) \in \wp_2(c_2, d_2)$:
- (O4.1) If $\gamma_1 \neq \gamma_2$, then $(p_1, \gamma_1) \in \wp(CR(c_1), CR(d_1))$ and $(p_2, \gamma_2) \in \wp(CR(c_1), CR(d_1))$.
 - (O4.2) If $\gamma_1 = \gamma_2$, then $\exists! p \in [min(p_1, p_2), max(p_1, p_2)]$ such that $(p, \gamma_1) \in \wp(CR(c_1), CR(d_1))$.
- This rule gives guidelines for solving conflicts between edges present in both ontologies; the interesting case occurs when two constraint probabilities from the same edge in the different CPOs have the same constraint. In such cases, the constraint appears in S with a probability value between the two probabilities on the conflicting edges.
- (O5) Every $c_1 \sqsubset_{(p_0, \text{true}), (p_1, \gamma_1), \dots, (p_k, \gamma_k)} c_2 \in \mathcal{I}$ induces an edge between $CR(c_1)$ and $CR(c_2)$ in $(\mathcal{C}, \Rightarrow)$, labeled with $(p_0, \text{true}), (p_1, \gamma_1), \dots, (p_k, \gamma_k)$. If the induced edge conflicts with another edge, then rule (O4) applies. Otherwise, the induced edge is part of the graph $(\mathcal{C}, \Rightarrow)$. Intuitively, this rule states that immediate subclassing constraints induce new edges in S and they are treated as standard edges in case of conflict.
- (O6) $\forall c \in \mathcal{C}_1 \cup \mathcal{C}_2, \forall \mathcal{L} \in \text{me}_{1|2}(c)$ (me_1 or me_2 is applied depending on where c is from), $\exists! \mathcal{L}' \in \text{me}(CR(c))$ such that $\mathcal{L} \subseteq \mathcal{L}'$. Intuitively, this rule tell us that a disjoint decomposition cannot be split as a result of merging.
- (O7) S is pseudoconsistent and well-structured according to Definitions 12 and 14. Intuitively, this rule states that the resulting CPO is consistent and this can be verified in polynomial time.

There can be zero, one or many witnesses to the integration of two CPOs. Two CPOs are said to be *unmergeable* if no witnesses exist. We now present an algorithm that takes two consistent CPOs and either returns a witness to the integration of the two if such a witness exists or throws an exception.

Algorithm: CPOmerge(S_1, S_2, \mathcal{I})

Input: $S_1 = (\mathcal{C}_1, \Rightarrow_1, \text{me}_1, \wp_1)$, $S_2 = (\mathcal{C}_2, \Rightarrow_2, \text{me}_2, \wp_2)$ – consistent CPOs, interoperation constraint set \mathcal{I} .

Output: $S = \mu_{\mathcal{I}}(S_1, S_2)$ or throws *cannot_merge_exception*.

Notation: We will use *union-find* semantics [4] for the first part of the algorithm. We will denote by $CR(x)$ the *union-find* class representative for element x and we will assume that *unify* is written such that $CR(x)$ will always be in \mathcal{C}_1 . We will regard $[\Rightarrow]$ and $[\text{me}]$ as data structures, although they are functions (the parentheses are used for notation purposes only). We will initialize these structures for each of the corresponding function's domain values.

```

(* INITIALIZATION *)
1)  $\mathcal{C} = \mathcal{C}_1$ ;
2)  $[\Rightarrow] = [\Rightarrow_1]$ ;
3)  $[me] = [me_1] \cup [me_2]$ ;
(* ADDING NODES FROM  $S_2$ *)
4) for each  $c_2 \in \mathcal{C}_2$  do
5)   if  $\exists (c_1 := c_2) \in \mathcal{I}$  then
6)     unify( $c_1, c_2$ );
7)   else
8)      $\mathcal{C} = \mathcal{C} \cup \{c_2\}$ ;
9)   end;
10) end;
(* CLEARING UP me *)
11) replace all classes  $x$  from all clusters within me with  $CR(x)$ ;
12) for each  $c \in \mathcal{C}$  do
13)   for each clusters  $\mathcal{L}_1, \mathcal{L}_2 \in me(c)$ ,  $\mathcal{L}_1 \neq \mathcal{L}_2$ , do
14)     for each  $(c_1, c_2) \in \mathcal{L}_1 \times \mathcal{L}_2$  do
15)       if  $CR(c_1) = CR(c_2)$  then
16)          $[me](c) = [me](c) - \{\mathcal{L}_1, \mathcal{L}_2\} \cup \{\mathcal{L}_1 \cup \mathcal{L}_2\}$ ;
17)         break;
18)       end;
19)     end;
20)   end;
21) end;
(* ADDING NEW EDGES *)
22) for each  $(c, d)$  edge in  $S_2$  or subclassing constraint in  $\mathcal{I}$  do
23)   if  $CR(c) \Rightarrow CR(d) \notin [\Rightarrow]$  then
24)      $[\Rightarrow] = [\Rightarrow] \cup (c, d)$ ;
25)     set  $\wp(c, d)$  to  $\wp_2(c, d)$  or the label of the subclassing constraint;
26)   else
27)      $e = [\Rightarrow](c, d)$ ;
28)      $e_1 =$  the second edge  $(c, d)$ 
28') (from subclassing constraint or  $S_2$ ).
29)   for each  $(p, \gamma) \in \wp_2^*(e_1)$  do
29') ( $\wp_2^*$  means either  $\wp_2$  or the subclassing constraint label)
30)     if  $\exists (p_1, \gamma) \in \wp(e)$  then
31)        $\wp(e) = \wp(e) - \{(p_1, \gamma)\} \cup \{(min(p, p_1), \gamma)\}$ ;
32)        $\wp_2^*(e_1) = \wp_2^* - \{(p, \gamma)\}$ ;
33)     end;
34)   end;
35)    $\wp(e) = \wp(e) \cup \wp_2^*$ ;
36)   end;
37) end;
38) if not Consistency-Check( $S$ ) then throw cannot_merge_exception;
39) return  $S$ ;

```

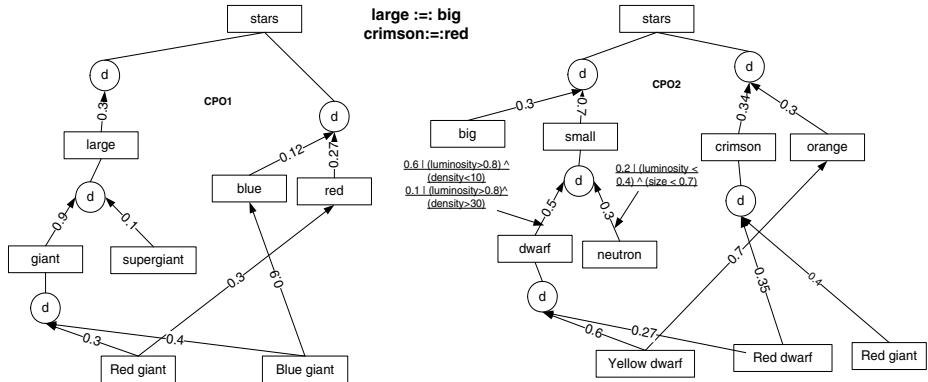


Fig. 2. Merging example

CPOmerge starts by using the class hierarchy of one of the two CPOs being merged. We can then use any of the numerous well-known *union-find* algorithms (Tarjan [4]) to add classes from the second ontology to the merge result. Since every class in the two ontologies appears at most once in the equality constraints, the size of each *union-find* set is at most 2, which means that there is almost no time penalty for using *union-find*.

The next step is needed to generate the new clustering function *me* for the resulting ontology. The property enforced here is: if we have two pairs of classes $c_1 := c_2$ and $d_1 := d_2$ and $c_1 \Rightarrow d_1$ and $c_2 \Rightarrow d_2$, then all siblings of c_2 need to be in the same disjoint decomposition as the siblings of c_1 . Note that this “cluster contraction” process does not need to be recursively repeated. If that were the case, we would be contracting different clusters from the same ontology, which would lead to an inconsistent merge result.

The next step implies adding the edges from the second ontology, as well as those implied by the subclassing constraints. Note that in case of conflict, we always take the minimum probability of the two constraint probabilities. Intuitively, this tends to minimize $w(c, d)$ and thus lead to a consistent merge result. The last step implies checking for consistency. If the resulting CPO is not consistent, then the merge operation fails. There are a number of techniques that can extend the merge operation to recover from inconsistencies, using loop elimination for instance. These are beyond the scope of this paper. Using the algorithm above, we can extend the merge operation to an arbitrary number of CPOs, by performing merging in pairs. The algorithm above is bounded by $\mathcal{O}(n^3)$, given by the loops involved in restructuring *me*. The two ontologies in Figure 2 can be merged into the ontology in Figure 1 using the rules depicted.

Theorem 4. (*CPOmerge correctness*).

- (i) $CPOmerge(S_1, S_2, \mathcal{I})$ returns a witness iff there exists a witness to the integrability of S_1, S_2 under interoperation constraints \mathcal{I} .
- (ii) *CPOmerge* is guaranteed to terminate.

The proof is based on the structure of the algorithm that directly corresponds to conditions $(O1)$ - $(O7)$. Also, the choice of probability in cases of conflict is conservative.

6 CPO Operations

We now extend the relational algebra to accommodate CPO-enhanced relations.

6.1 Selection

Definition 17. (*Simple condition*). Suppose \mathcal{C} is the set of classes in the CPO associated with relation \mathcal{R} with attributes $\{A_1, \dots, A_m\}$. Then: a simple condition is of the form $A_i \text{ in } T$, where $T \subseteq \text{dom}(A_i) \subseteq \mathcal{C}$.

Example 3. The definition states that in the context of a CPO, simple conditions are those that refer to the attributes represented in the CPO. This is very similar to Definition 3 for simple constraints. However, simple constraints can refer to all attributes of a relation *except* those associated with the current CPO, while a simple condition refers *only* to the attributes associated with the CPO. For instance, in the example of Figure 1, $\text{luminosity} > 0.8$ is a simple constraint, but not a simple condition since the CPO does not contain values for the luminosity attribute. On the other hand, $\text{star_type} \text{ in } \{\text{dwarf}, \text{orange}\}$ is a simple condition, but cannot be a simple constraint, since it refers to the attribute associated with the CPO.

Definition 18. (*Probability condition*). A probability condition for selection is of the form (δ, p) , where δ is a simple condition and p is a rational number in the $[0, 1]$ interval.

Intuitively, a probability condition ensures that only those tuples which satisfy δ with probability p or more will be returned.

Definition 19. (*Selection condition*). A selection condition is any combination of probability conditions or simple conditions using the \vee , \wedge and \neg logical operators.

Example 4. Intuitively, the selection conditions we are interested in are combinations of simple and probability conditions and thus refer only to the attributes associated with a CPO. All other selection operations can be reduced to classical relational algebra. For example,

$$(\text{star_type} \text{ in } \{\text{dwarf}, \text{giant}\}, 0.5) \wedge \neg(\text{star_type} \text{ in } \{\text{blue}\})$$

is a selection condition.

We now give an extension for the relational selection operation. This definition uses the concept of probability path defined in Section 3.

Definition 20. For a CPO-enhanced relation defined by the CPO $S = (\mathcal{C}, \Rightarrow, me, \wp)$ and the relation $\mathcal{R} = \{A_1, \dots, A_m\}$, we define CPO-enhanced selection as follows (Note: As stated above, we are only interested in cases not covered by relational algebra's selection).

- (*Simple selection*). We define

$$\sigma_{A_i \in T}(\mathcal{R}) = \{t \in \mathcal{R} | t.A_i \in T \vee \exists c \in T \text{ s.t. } t.A_i \Rightarrow^* c\}.$$

- (*Probabilistic selection*). We define $\sigma_{(A_i \in T, p)}(\mathcal{R}) = \{t \in \mathcal{R} | (t.A_i \in T) \vee (\exists c \in T \text{ s.t. } t.A_i \Rightarrow^* c) \vee (\exists c' \in T \text{ s.t. } c' \rightsquigarrow_p t.A_i \text{ and } t.A_i \text{ meets every constraint in } \Gamma(c' \rightsquigarrow_p t.A_i))\}$.
- (*Composite selection*). For every selection condition of the form $\Delta = \Delta_1 \text{ op } \Delta_2$, $\sigma_\Delta(\mathcal{R}) = \sigma_{\Delta_1}(\mathcal{R}) \overline{\text{op}} \sigma_{\Delta_2}(\mathcal{R})$. Here, the $\overline{\text{op}}$ operator is the set operation corresponding to the logical operation op . For instance, \cup corresponds to \vee , \cap to \wedge and so on.

Example 5. In the examples of Section 1, the query $\sigma_{(\text{star_type} \in \{\text{dwarf}\}, 0.5)}(\text{Stars})$ would return:

- All tuples with $\text{star_type} = \text{dwarf}$;
- All tuples with $\text{star_type} \in \{\text{yellow_dwarf}, \text{red_dwarf}\}$;
- All tuples with $\text{star_type} = \text{small}$. If the probability threshold had been 42%, then we would have also included all tuples that have $\text{star_type} = \text{star}$ and meet the conditions $\text{luminosity} > 0.8$ and $\text{density} < 10$.

6.2 Projection and Cartesian Product

Projection for a CPO-enhanced relation is the same as in relational algebra. An operation that would “trim” the CPO graph is possible, but that would lead to loss of data within the relation. A cartesian product operation on two CPO-enhanced relations will not alter the structure of the respective CPOs, nor the association between the CPOs and the attributes of the two relations. Thus, we reduce the cartesian product operation to its correspondent in classical relational algebra.

6.3 Join

Consider two CPO-enhanced relations $(\mathcal{T}_1, \mathcal{R}_1)$ and $(\mathcal{T}_2, \mathcal{R}_2)$ with mapping functions f_1 and f_2 respectively and suppose their attribute names are renamed so that they share no attributes in common. Suppose $\xi(\mathcal{A}_1, \mathcal{A}_2)$ is a join-condition linking attribute A_1 from \mathcal{R}_1 with attribute A_2 from \mathcal{R}_2 . We say that $(\mathcal{T}_1, \mathcal{R}_1)$ and $(\mathcal{T}_2, \mathcal{R}_2)$ are *join compatible* w.r.t. a finite set \mathcal{I} of interoperability constraints iff the CPOs $f_1(A_1)$ and $f_2(A_2)$ are mergeable w.r.t. \mathcal{I} and join-condition $\xi(\mathcal{A}_1, \mathcal{A}_2)$.

Definition 21. (*Join operation*). Suppose $(\mathcal{T}_1, \mathcal{R}_1)$ and $(\mathcal{T}_2, \mathcal{R}_2)$ are join-compatible w.r.t. \mathcal{I} and the join-condition $\xi(\mathcal{A}_1, \mathcal{A}_2)$. Let S be a witness to the integrability of the ontologies associated with $\mathcal{A}_1, \mathcal{A}_2$. Then

Table 1. Simple relation examples for join

Relation $R1$			Relation $R2$		
Name	Density	Star_type	Mass	Location	Star_type
Gliese623a	0.9	red	0.1	Hercules	crimson
Gliese623b	1.2	red_dwarf	143	Leo	blue_giant

- $\mathcal{R} = \mathcal{R}_1 \bowtie_{\xi(\mathcal{A}_1, \mathcal{A}_2)} \mathcal{R}_2$
- $\mathcal{T} = (\mathcal{T}_1 \cup \mathcal{T}_2 \cup \{S\}) - \{S_1, S_2\}$ where S_i is the ontology associated with \mathcal{A}_i .

Table 2. Merge result for $(CPO1, R1) \bowtie_{R1.\text{Star_type}=R2.\text{Star_type}} (CPO2, R2)$

Name	Density	R1.Star_type	Mass	Location	R2.Star_type
Gliese623a	0.9	red	0.1	Hercules	crimson

Example 6. Given the relations $R1$ and $R2$ in Table 1, assume that the attribute $R1.\text{Star_type}$ is mapped to $CPO1$ in Figure 2 and that attribute $R2.\text{Star_type}$ is mapped to $CPO2$ in the same figure. Also, we denote by $\mathcal{I}(S1, S2)$ the set of interoperation constraints for the purpose of merging $S1$ and $S2$. \mathcal{I} will contain the equality constraints depicted in Figure 2, as well as any other constraints based on class names that are identical in the two ontologies. Then the result of the join operation $(CPO1, R1) \bowtie_{\text{Star_type}} (CPO2, R2)$ is presented in Table 2, with the $R1.\text{Star_type}$ and $R2.\text{Star_type}$ being mapped to the CPO presented in Figure 1 (here we abbreviate the condition that the Star_type attributes of the two relations should be equal). We obtain this result by first performing the merge between $CPO1$ and $CPO2$ and then checking for the join condition, taking into account any equality constraints inferred.

6.4 Union

Definition 22. (*Union operation*). Suppose $(\mathcal{T}_1, \mathcal{R}_1)$ and $(\mathcal{T}_2, \mathcal{R}_2)$ are CPO-enhanced relations with the same schema. For each attribute A_i , let S_i^1, S_i^2 be the CPO associated with attribute A_i in $(\mathcal{T}_1, \mathcal{R}_1)$ and $(\mathcal{T}_2, \mathcal{R}_2)$ respectively. Let I_i be a set of interoperation constraints. We say $(\mathcal{T}_2, \mathcal{R}_2)$ are CPO-union compatible if S_i^1, S_i^2 have a witness S_i to their integrability under I_i . In this case, we define $(\mathcal{T}_1, \mathcal{R}_1) \cup (\mathcal{T}_2, \mathcal{R}_2)$ as $(\mathcal{T}, \mathcal{R})$ where:

- $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ Here, \cup denotes the relational algebra union operation.
- $\mathcal{T} = \{S_i \mid A_i \text{ is an attribute of } \mathcal{R}_i\}$.

6.5 Intersection and Difference

In order to perform intersection and difference in relational algebra, the relations involved must be union compatible. Therefore, the intersection and difference operations for CPO-enhanced relations are defined similarly to the union operation. The resulting relation \mathcal{R} is either $\mathcal{R}_1 - \mathcal{R}_2$ for difference or $\mathcal{R}_1 \cap \mathcal{R}_2$ for intersection.

7 Related Work and Conclusions

Wiederhold's group was the first to notice that ontologies can be used to improve the quality of answers to queries. They proposed an *ontology algebra* [5, 6, 7]. Their

algebras consisted of logical statements [5] using a LISP style syntax. Their later work included graphs which were combined in their ONION system using the three set operations of union, intersection and difference. In addition, a framework called TOSS [8] was recently proposed to increase the recall of queries to XML databases.

In this paper, we present the concept of a probabilistic ontology that builds on the work of Eiter et. al. [1, 3]. We extend the concept of a POB-schema by Eiter et. al. to the case of a constrained probabilistic ontology (CPO) and show how the consistency of POB-schemas can be extended to CPOs. However, our path from there is very different. We show how the answers to *relational queries* can be greatly improved by using probabilities *within the system hidden from users*. We show how multiple CPOs can be combined under a variety of interoperation constraints. Based on these ideas, we develop a relational-style algebra to handle querying with associated ontologies. The interested reader may find initial experimental results, as well as a discussion on how CPOs can be inferred from data sources using a number of different methods in the extended version of the paper available at <http://www.cs.umd.edu/users/udrea/ProbOntologies.pdf>.

Acknowledgements

Work supported in part by ARO grant DAAD190310202, NSF grants IIS0329851 and 0205489, and by a DARPA subcontract from the Univ. of California Berkeley.

References

1. Eiter, T., Lu, J.J., Lukasiewicz, T., Subrahmanian, V.: Probabilistic object bases. *ACM Trans. Database Syst.* **26** (2001) 264–312
2. Fagin, R., Halpern, J.Y., Megiddo, N.: A logic for reasoning about probabilities. *Inf. Comput.* **87** (1990) 78–128
3. Eiter, T., Lukasiewicz, T., Walter, M.: A data model and algebra for probabilistic complex values. *Ann. Math. Artif. Intell.* **33** (2001) 105–252
4. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. *J. ACM* **22** (1975) 215–225
5. Maluf, D., Wiederhold, G.: Abstraction of representation for interoperation. *Lecture Notes in AI*, subseries of *LNCS* **1315** (1997)
6. Mitra, P., Wiederhold, G., Kersten, M.: A graph-oriented model for articulation of ontology interdependencies. In: *Proceedings Conference on Extending Database Technology*, (EDBT’2000). (2000) 303–316
7. Wiederhold, G.: Interoperation, mediation and ontologies. In: *International Symp. on Fifth Generation Computer Systems, Workshop on Heterogeneous Cooperative Knowledge Bases*. (1994) 33–48
8. Hung, E., Deng, Y., Subrahmanian, V.S.: Toss: an extension of tax with ontologies and similarity queries. In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, ACM Press (2004) 719–730

Intelligent Web Service - From Web Services to .Plug&Play. Service Integration

Erich Neuhold, Thomas Risse, Andreas Wombacher,
Claudia Niederée, and Bendick Mahleko

Fraunhofer Institute (IPSI), Germany

Abstract. The service oriented architecture and its implementation by Web services have reached a considerable degree of maturity and also a wide adoption in different application domains. This is true for the R&D as well as for the industrial community. Standards for the description, activation, and combination of Web services have been established; UDDI registries are in place for the management of services, and development environments support the software engineer in the creation of Web services.

However, the major benefit of service oriented architectures, the loose coupling of services, is still seldom explored in real world settings. The reason is the heterogeneity on different levels within the service oriented architecture. The heterogeneity problems reach from the semantics of service descriptions to compatibility problems between workflows, which have to be connected via service interfaces. In spite of compatible service signatures, workflows might, for example, not be compatible in their semantics.

This talk discusses challenges and solutions for a real .Plug&Play. service infrastructure, i.e. a Web service infrastructure, where integration of new Web services becomes as simple and straightforward as plugging a USB stick into your laptop. To achieve this goal various issues have to be addressed:

- Semantics of services as a foundation for intelligent service mediation and usage
- Effective, automatic, and intelligent service discovery taking into account application context
- Dynamic context-aware composition of services into processes

The challenges and approaches for a “Plug&Play” service infrastructure are illustrated with a real world example.

Brief Speaker Bio

Erich Neuhold received his M. S. in Electronics and his Ph.D. degree in Mathematics and Computer Science at the Technical University of Vienna, Austria, in 1963 and 1967, respectively. Since 1986 he has been Director of the Institute for Integrated Publication and Information Systems (IPSI) in Darmstadt, Germany (a former Institute of the German National Research Center for Information

Technology - GMD, since July 2001 a Fraunhofer Institute). He is a member of many professional societies, an IEEE senior member, and currently holds the chairs of the IEEE-CS the Technical Committee on Data Engineering.

Dr. Neuhold is also Professor of Computer Science, Integrated Publication and Information Systems, at the Darmstadt University of Technology, Germany. His primary research and development interests are in heterogeneous multimedia database systems in Peer-to-Peer and GRID environments, WEB technologies and persistent information and knowledge repositories (XML, RDF, ...) and content engineering. In content engineering special emphasis is given to all technological aspects of the publishing value chain that arise for digital products in the WEB context. Search, access and delivery includes semantic based retrieval of multimedia documents. He also guides research and development in user interfaces including virtual reality concepts for information visualization, computer supported cooperative work, ambient intelligence, mobile and wireless technology, security in the WEB and applications like e-learning, e-commerce, e-culture and e-government.

Process Modeling in Web Applications

Stefano Ceri

Dipartimento di Elettronica e Informazione,
Politecnico di Milano, Italy

Abstract. While Web applications evolve towards ubiquitous, enterprise-wide or multi-enterprise information systems, their features must cover new requirements, such as the capability of managing complex processes spanning multiple users and organizations, by interconnecting software provided by different organizations. Significant efforts are currently being invested in application integration, to support the composition of business processes of different companies, so as to create complex, multi-party business scenarios. In this setting, Web applications, which were originally conceived to allow the user-to-system dialogue, are extended with Web services, which enable system-to-system interaction, and with process control primitives, which permit the implementation of the required business constraints. This talk presents new Web engineering methods for the high-level specification of applications featuring business processes and remote services invocation. Process- and service-enabled Web applications benefit from the high-level modeling and automatic code generation techniques that have been fruitfully applied to conventional Web applications, broadening the class of Web applications that take advantages of these powerful software engineering techniques. All the concepts presented in this talk are fully implemented within a CASE tool.

CoopIS 2005 PC Co-chairs' Message

This volume contains the proceedings of the Thirteenth International Conference on Cooperative Information Systems, CoopIS 2005, held in Agia Napa, Cyprus, November 2 - 4, 2005.

CoopIS is the premier conference for researchers and practitioners concerned with the vital task of providing easy, flexible, and intuitive access and organization of information systems for every type of need. This conference draws on several research areas, including CSCW, Internet data management, electronic commerce, human-computer interaction, workflow management, web services, agent technologies, and software architectures.

These proceedings contain 33 original papers out of 137 submissions. Papers went through a rigorous reviewing process (3 reviewers per paper) and were sometimes discussed by email. The papers cover the fields of Workflows, Web Services, Peer-to-Peer interaction, Semantics, Querying, Security, Mining, Clustering, and Integrity. We wish to thank the authors for their excellent papers, the Program Committee members, and the referees for their effort. They all made the success of CoopIS 2005 possible.

August 2005

Mohand-Said Hacid, Université Claude Bernard Lyon 1
John Mylopoulos, University of Toronto
Barbara Pernici, Politecnico di Milano
(CoopIS 2005 Program Committee Co-Chairs)

Let's Go All the Way: From Requirements Via Colored Workflow Nets to a BPEL Implementation of a New Bank System

W.M.P. van der Aalst^{1,2}, J.B. Jørgensen², and K.B. Lassen²

¹ Department of Technology Management, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl

² Department of Computer Science, University of Aarhus,
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark
{bjb, krell}@daimi.au.dk

Abstract. This paper describes use of the formal modeling language *Colored Petri Nets (CPNs)* in the development of a new bank system. As a basis for the paper, we present a *requirements model*, in the form of a CPN, which describes a new bank work process that must be supported by the new system. This model has been used to specify, validate, and elicit user requirements. The contribution of this paper is to describe two translation steps that go from the requirements CPN to an implementation of the new system. In the first translation step, a *workflow model* is derived from the requirements model. This model is represented in terms of a so-called *Colored Workflow Net (CWN)*, which is a generalization of the classical workflow nets to CPN. In the second translation step, the CWN is translated into implementation code. The target implementation language is BPEL4WS deployed in the context of IBM WebSphere. A semi-automatic translation of the workflow model to BPEL4WS is possible because of the structural requirements imposed on CWNs.

Keywords: Business Process Management, Workflow Management, BPEL4WS, Colored Petri Nets.

1 Introduction

Bankdata is a Danish company that is currently developing a new system called the *Adviser Portal (AP)*. AP has been bought by 15 Danish banks and will be used by thousands of bank advisers in hundreds of bank branches. The scope of the system is to support advising private customers and small businesses. The total development effort is 15 developers over a period three years. The first version is planned to become operational in September 2005.

The main goal of AP is to increase the efficiency and quality of bank advisers' work. Currently, prior to the deployment of AP, the advisers in Bankdata's customer banks often need information, which is scattered over many places: in different IT systems, on paper sheets in binders or in piles on a desk, on post-it notes, or even only in the minds of advisers. This hampers both efficiency

and quality; it is time-consuming to search for information, and an adviser may, e.g., sometimes forget to call a customer when she has promised to do so. The scattering of information makes it difficult for an adviser to get an overview, both of her own current and future tasks, and of the information pertaining to a particular task. Moreover, it makes it difficult for the bank, as an organization, to coordinate, distribute, and plan work.

Problems like these are mainly caused by the nature of the bank advisers work processes. To overcome the problems, the AP system will represent a change in perspective for Bankdata's software development. Previously, Bankdata focused on the development of traditional data-centric systems. For the new AP system, Bankdata uses a process-centric approach: In the new system, there is more focus on the work processes that must be supported than there has been previously. Thus, a workflow management system is a central component of AP.

Our focus in this paper is on AP's support of one specific work process: The process describing how bank advisers give advice to customers enquiring about getting a so-called *blanc loan*. A blanc loan is a simple type of loan, which can be granted without requiring the customer to provide any security. This is in contrast to, e.g., mortgage credits and car loans. Blanc loans are typically used for consumption purposes like travels, weddings, and gifts. They constitute a relatively high risk for the banks and have a correspondingly high interest rate.

We will describe the use of the formal modeling language *Colored Petri Nets (CPNs)* [16, 20] in the development of AP. First, CPN has been used as a vehicle for requirements engineering for AP. This involved using a *requirements model* in the form of a CPN as the core ingredient of an *Executable Use Case (EUC)* [17] to describe new work processes and their proposed computer support. This has been a means to specify, validate, and elicit requirements in a number of workshops with future users and systems analysts from Bankdata. We will present the *Requirements CPN (RCPN)*, and we will briefly outline how it has been used. However, the main focus of this paper is on two translation steps taken to close the gap between the requirements model and the implementation of the new system. The first step translates the RCPN into a *workflow model* in the form of a *Colored Workflow Net (CWN)*, a new class of Petri nets that we will introduce. The second step translates the CWN into the chosen implementation language, which is *Business Process Execution Language for Web Services (BPEL4WS)* [5]. (In this paper we will simply use "BPEL" to refer to this de-facto standard.)

The CPNs we present in this paper are created using *CPN Tools*, a graphical environment to model, enact and analyze CPNs. (CPN Tools can be downloaded from www.daimi.au.dk/CPNtools/.)

This paper is structured as follows. Section 2 introduces the overall approach, including the two translation steps in focus. Section 3 presents the requirements model. Section 4 introduces the CWN modeling language and describes how the RCPN is translated into a workflow model in the form of a CWN. Section 5 discusses how the CWN is translated into BPEL. Section 6 discusses related work. Section 7 concludes the paper by summarizing the main results and discussing future work.

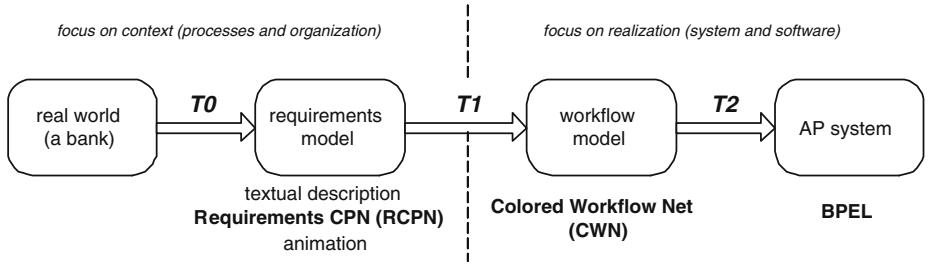


Fig. 1. Overview of the different models and translations between these models

2 Overall Approach

Figure 1 summarizes the overall approach. Translation T_0 creates a model of the real world, in this case the processes and people within banks. The result of T_0 is the *requirements model*. In the technical report [18], we describe how T_0 is made in cooperation between users and Bankdata analysts, and how the requirements model is used in an iterative, prototyping fashion; it is constructed based on informal textual descriptions and diagrams and has served as an engine to drive a graphical animation.

Translation T_1 derives the *workflow model* from the requirements model. The latter includes both: (1) actions that are to remain manual, when the new system is deployed; (2) actions that will be supported by the new system in interaction with human users; (3) actions that are to be fully automated by the new system. The workflow model includes only actions in categories (2) and (3). Moreover, the workflow model adds more details. The requirements model uses the CPN language in an unrestricted manner, e.g., tokens, places, and transitions may represent any entity deemed to be relevant. In the workflow model, we restrict ourselves to use only concepts and entities, which are common in workflow languages. More specifically, we propose *Colored Workflow Nets (CWNs)* as the language for making workflow models. A CWN is a CPN model restricted to the workflow domain and can be seen a high-level version of the traditional *Workflow Nets* (WF-nets) [1].

Translation T_2 goes from the CWN into skeleton code for the chosen implementation platform. AP is implemented using the IBM WebSphere platform; IBM WebSphere includes the workflow management tool IBM Process Choreographer, which will be used to orchestrate some of the work processes that are currently carried out manually in the banks. IBM Process Choreographer uses BPEL. Therefore, translation T_2 translates the CWN into BPEL.

As shown in Figure 1 by a dashed line, translation T_1 represents a shift in focus. Left of T_1 , the focus is on the *context* (processes and organization). Right of T_1 , the focus is on the *realization* (system and software). The use of CPN as a common language for both the requirements model and the workflow model provides a natural link of these two views. This facilitates a smooth transition and is an attempt to avoid the classical “disconnect” between business processes and IT.

All translations T_0 , T_1 and T_2 in Figure 1 are done manually for the AP system in consideration. Both T_0 and T_1 are likely to remain manual given their characteristics; they inherently involve human analysis, decisions, and agreements between stakeholders. Translation T_2 , however, can be supported using a computer tool that generates template code for the chosen implementation platform. We have developed a systematic approach to transform a CWN into BPEL code. This approach is semi-automatic, i.e., the template code is generated on the basis of the structure of the underlying workflow net, as described in the technical report [4].

The translations we present should be seen as a proof-of-concept: They do not yield a full implementation of the AP system, but they merely demonstrate the viability of our approach.

3 Requirements Model

In this section, we first present the requirements model, i.e., the RCPN. The RCPN has been used as an ingredient of an Executable Use Case (EUC) [17], which support specification, validation, and elicitation of requirements. EUCs spur communication between stakeholders and can be used to narrow the gap between informal ideas about requirements and the formalization that eventually emerges when a system is implemented. An EUC may be seen as a *context-descriptive prototype* [6]. In this way, the RCPN has played a similar role as a high-fidelity prototype implemented in a programming language.

The use of EUCs in the development of AP is described in [18] (this is translation T_0 of Figure 1). In the present paper, we merely give an impression of the EUC by showing the animation, which is part of it. Figure 2 mimics a situation in a bank in which there are two advisers, Ann and Bill, their manager Mr. Banks, and one customer, Mr. Smith. The circles represent blanc loan enquiries. The anima-

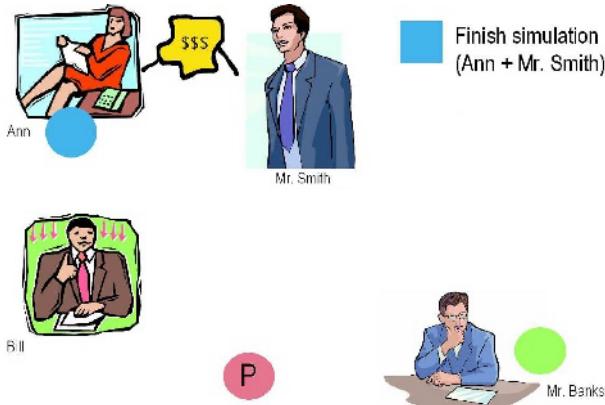


Fig. 2. Snapshot of graphical animation

tion user can play with the animation and try out various scenarios and carry out experiments. Note that the animation is constructed using the animation facilities offered on top of CPN Tools and that the animated objects interact directly with the running CPN model.

We now present the RCPN; an extract is shown in Figure 3. At the same time, we also give an informal primer to CPN, which allows the reader to understand the CPNs in general terms. The primer does not account for all the technicalities of CPN; its purpose is only to provide an overall impression of the most basic concepts of the CPN language. For a more thorough introduction, the reader is referred to [16, 20].

A CPN describes a system's states and actions. The state of a system is modeled by *places*, drawn as ellipses. Places can hold *tokens*, which have different types or colors in CPN jargon. These types are a subset of the data types in Standard ML such as the primitive types integer and string and compositional types such as tuple, list and record. Each place can hold tokens of a certain type. Usually the type of a place is written in capital letters close to it. All places in this CPN model have the type LOAN. Figure 4 shows the definition of LOAN.

LOAN is a record type; it consists of a *caseid* for separating different loan case instances, a *customer*, a *responsible* person (either an adviser or a manager), the *status* of the loan (ongoing, grant, etc.), the *amount* to loan, the *monthlyFee* to pay for the loan with a given loan setup, the *interestRate*, the *duration* for paying back the loan, the *purpose* for loaning the money and the *account* to put the money into.

A CPN's actions are represented by *transitions*, which are drawn as rectangles. *Arcs* connect transitions and places. An arc can only connect a transition with a place or vice versa; it is not possible to e.g. connect two places to each other.

A CPN can be executed, i.e., there is a semantics specifying when a transition is *enabled* and what happens when the transition *occur*. A transition is enabled if certain tokens required for the action are present on the transition's input places. The input arcs to the transition describe which tokens are needed for the transition to be enabled. For example, for transition **Lookup customer information and credit information** to occur, a token of type LOAN must be present on the place **Customer observed**.

When a transition occurs, all tokens that are needed for its enabling are consumed and tokens are produced on all output places as described by the outgoing arcs. For example, when transition **Lookup customer information and credit information** occurs, a token of type LOAN is consumed from **Customer observed** and depending on whether the status of the LOAN token is set to **refusal** or **ongoing**, the token will be moved to either **Early refusal** or **Ready for advising**.

A CPN may consist of multiple modules, organized in a hierarchy. The RCPN consists of 7 modules and the module shown in Figure 3 is the top level model. As placeholders for modules on lower levels, so-called substitution transitions are used. A substitution transition is represented as a rectangle with a small box with the module name near it. For example, **Advising / Simulation** is a substitution transition. This means that the details of how advising and what is called simulation is done are modeled on another module of the model. Note that in the jargon used

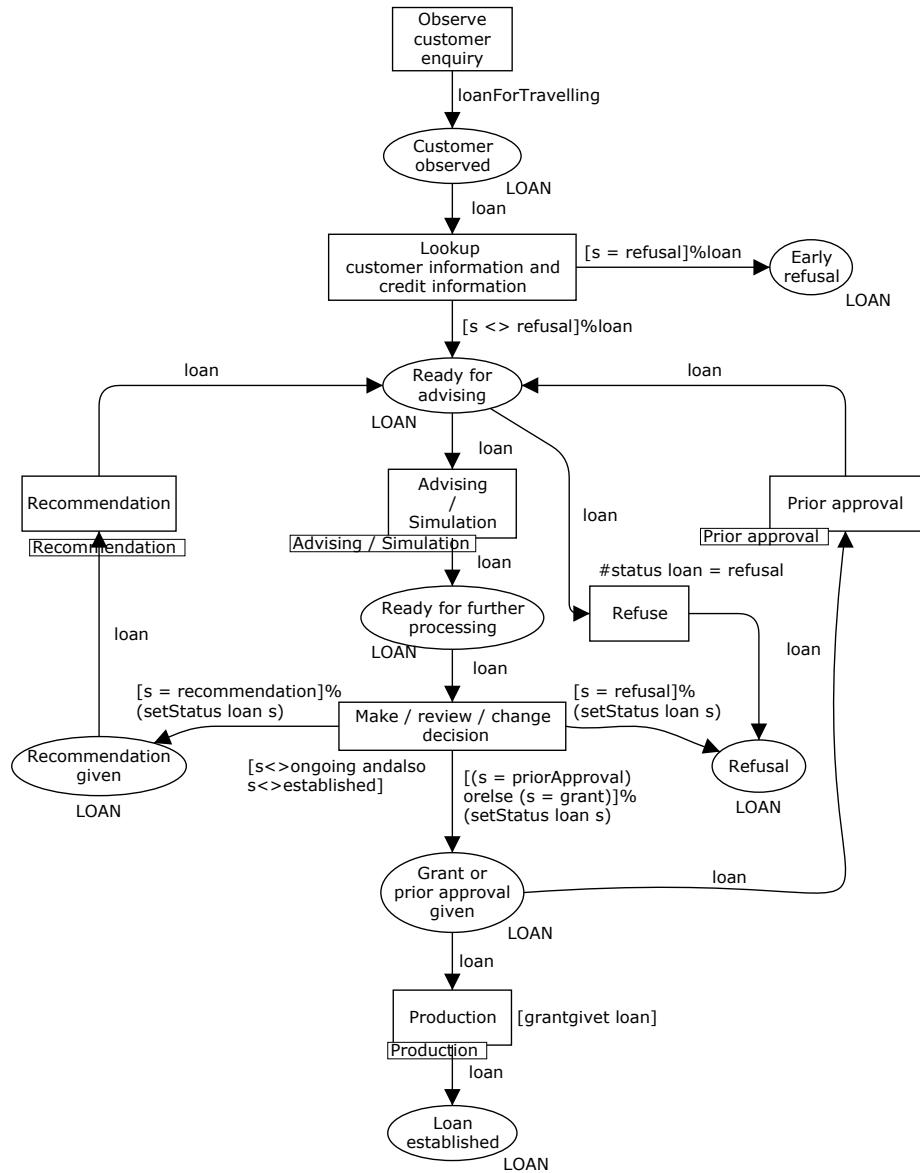


Fig. 3. Extract of the RCPN

in the banks, *simulation* means that an adviser does some calculations and suggests various values for monthly payment, interest rate, and loan period to a customer.

The RCPN describes the control flow of actions which can happen from the point when a customer makes a blanc loan application until the request is either

```

colset LOAN = record caseid: INT
    * customer: STRING
    * responsible: STRING
    * status: STATUS
    * amount: AMOUNT
    * monthlyFee: MONTHLYFEE
    * interestRate: INTEREST
    * duration: DURATION
    * purpose: STRING
    * account: INT;

```

Fig. 4. LOAN type

refused or established and in what sequence actions can occur. In the following, we describe how a single loan application is handled.

When transition **Observer customer enquiry** occurs, a LOAN token is put on **Customer observed**. Two things can happen at this point: (1) The adviser refuses the loan right away, e.g. if he knows that the customer has a bad credit history; (2) the adviser agrees to handle the loan application. In (1), the blanc loan application is terminated and a LOAN token is put on the place **Early refusal**. In (2), a LOAN token is put on **Ready for advising**. When the loan is in the state **Ready for advising**, it can go into the module **Advising / Simulation**. This is a module which models the creation of a task in a advisers task list and the first blanc loan application setup; i.e. how much money to loan, at which interest rate and so on. After this is done the LOAN token is put on **Ready for further processing**.

A choice is made at transition **Make / review / change decision**. This models the choice which the adviser must take at this point. The choices are *grant*, *recommendation*, *prior approval* or *refusal*, which are explained below.

- *Grant* is given if the blanc loan application is reasonable this choice is made. The LOAN token will be moved to **Grant or prior approval given** and then into the **Production** module. Here the blanc loan is finalized. This involves printing some necessary documents. Finally, the token is placed on the place **Loan established**.
- *Recommendation* means that the adviser’s manager must make the decision. E.g., advisers are only permitted to grant loans up to a certain amount of money. This is modeled by the LOAN token being moved to **Recommendation given**. After this point it can enter the **Recommendation** module in which the behavior of the manager changing the status of the loan to either grant or refusal is modeled. After the module has executed, the token is moved to **Ready for advising**. The behavior described earlier from this point can occur again.
- *Prior approval* is the case where the adviser accepts the blanc loan application but needs more information from the customer before continuing to process the blanc loan. Here the token is moved to **Grant or prior approval given**. From here it can enter the module **Prior approval** which models the situation where the processing of the loan is postponed. After this has been executed, the LOAN

token is moved to **Ready for advising**. If a blanc loan has been given a prior approval it will always be granted at some point.

- *Refusal* applies if the adviser and customer cannot agree on a loan setup the application is refused. This is modeled by moving the **LOAN** token to **Refusal**. Then no other activities are possible for that particular loan application; this models that the case is ended.

All runs of a blanc loan application either end up in the state **Early refusal**, **Refusal**, or **Loan established**. The first two are for cases where applications are refused and the third is for cases where applications are approved.

4 From Requirements Model to Workflow Model

In this section, we address translation *T1* of Figure 1, which takes the RCPN described in the previous section and transforms it into a workflow model represented in terms of a CWN. We first motivate the need for a workflow model, then we introduce the class of CWNs, and finally, we present the CWN model for our case study.

4.1 Requirements Models Versus Workflow Models

In an RCPN, tokens, places and transitions may be used to represent arbitrary concepts relevant for requirements engineering. For example, tokens are used to represent customers, meeting rooms, conflicts, office equipment, paper document, etc. Some of these concepts have no counterpart in the final system. Given that a workflow management system is to be used, the vocabulary must be limited to the concepts supported by that system. For example, a workflow management system may know about cases, case attributes, tasks, roles, etc. but may not support concepts like meeting rooms and conflicts. Therefore, we propose a *system-independent language in-between the requirements level and the implementation level*. To do this, we first define some standard terminology and discuss differences between the requirements-level and workflow-level models.

Workflow processes, like the processing of blanc loans, are *case-driven*, i.e., tasks are executed for specific cases. A *case* may represent a blanc loan, but also the request to open a bank account, a customer complaint, or an insurance claim. These case-driven processes, also called *workflows*, are marked by at least the following three dimensions: (1) the *control-flow* dimension, (2) the *resource* dimension, and (3) the *case* dimension [1]. The control-flow dimension is concerned with the partial ordering of *tasks*. The tasks which need to be executed are identified and the routing of cases along these tasks is determined. Conditional, sequential, parallel and iterative routing are typical structures specified in the control-flow dimension. Tasks are executed by *resources*. Resources are human (e.g., an adviser) and/or non-human (e.g., a printer). In the resource dimension these resources are classified by identifying *roles* (resource classes based on functional characteristics) and *organizational units* (groups, teams or departments). Each resource may have multiple roles and belong to multiple organizational units. For the execution of a task,

a collection of resources may be required. (Although most workflow management systems assume only one resource to be involved in the execution of a task.) The required properties of the resources involved may be defined in terms of roles and organizational units, e.g., task “Recommend” needs to be executed by a “manager” of the “loans department”. Both the control-flow dimension and the resource dimension are generic, i.e., they are not tailored towards a specific case. The third dimension of a workflow is concerned with individual cases which are executed according to the process definition (first dimension) by the proper resources (second dimension). Cases may have attributes, e.g., some account number or the interest rate. The attribute names and their types are generic while the concrete attribute values are specific for a concrete case.

In summary, workflow models should have the following three properties:

- W1** The workflow model should use a *restricted vocabulary* common for workflow management systems as indicated above. Only concepts such as case, task, resource, role, organizational unit, and attribute may be used to construct workflow models.
- W2** The workflow model includes *only actions* (i.e., tasks) which are to be *supported by the workflow management system* (i.e., it includes no tasks, e.g. manual actions, the system will not be aware of).
- W3** The workflow model *refines* selected parts of the requirements model to enable system support. Note that granularity of tasks may not be dealt with in detail at the requirements-level. However, at the workflow-level, the splitting and/or merging tasks is important as it directly influences the implementation of the system.

This paper proposes CWNs as a workflow modeling language sitting in-between the RCPN and BPEL, as shown in Figure 1.

4.2 Colored Workflow Nets

A *Colored Workflow Net* (CWN) is a CPN as defined in [16, 20]. However, it is restricted as indicated in Section 4.1. Note that a CWN covers the *control-flow* perspective, the *resource* perspective, and the *data/case* perspective, and abstracts from implementation details and language/application specific issues. A CWN should be a CPN with only places of type **Case**, **Resource** or **CxR**. These types are as defined in Figure 5.

A token in a place of type **Case** refers to a case and some or all of its attributes. Tokens in a place of type **Resource** represent resources. Places of type **CxR** hold tokens that refer to both a case and a resource. Such places are used if resources need to execute a sequence of tasks for the same case, e.g., chained execution.

A CWN where all places of type **Resource** are removed should correspond to a *Sound Workflow Net* (sound WF-net) as defined in [1]. Although WF-nets have been defined for classical Petri nets it is easy to generalize the definition to CPN as discussed in [1, 2]. The basic requirement is that there is one source place and one sink place and all other nodes (places and transitions) are on a path from source

```

colset CaseID =union C:INT;
colset AttName = string;
colset AttValue = string;
colset Attribute = product AttName * AttValue;
colset Attributes = list Attribute;
colset Case = product CaseID * Attributes timed;
colset ResourceID = union R:INT;
colset Role = string;
colset Roles = list Role;
colset OrgUnit = string;
colset OrgUnits = list OrgUnit;
colset Resource = product ResourceID * Roles * OrgUnits timed;
colset CxR = product Case * Resource timed;

```

Fig. 5. Places in a CWN need to be of type **Case**, **Resource** or **CxR**

to sink. Moreover, given a token on the input place, eventually one token should appear on the output place and the rest of the places should be empty.

There should be *conservation of cases and resources*. This can be formulated in terms of colored place invariants [16]. For every resource place r there should be a (colored) place invariant associating weight 1 to r and selected places of type **CxR** (if any) and weight 0 to all other places. Moreover, there is a (colored) place invariant associating weight 1 to the source and sink place and positive weights to all other places of type **Case** or **CxR** (and weight 0 to all places of type **Resource**).

Transitions correspond to tasks supported by the workflow system. They should not violate the soundness and conservation properties mentioned above. Therefore, we propose the following *guidelines*.

- The following variables should be defined: c, c_1, c_2, c_3 , etc. of type **Case** and r, r_1, r_2, r_3 , etc. of type **Resource**. Using a fixed set of variables facilitates both the interpretation and automated translation of CWNs. For similar reasons we structure the arc inscriptions and guards (see below).
 - c, c_1, c_2, c_3 , etc. for arcs from places of type **Case**.
 - r, r_1, r_2, r_3 , etc. for arcs from places of type **Resource**.
 - $(c, r), (c_1, r_1), (c_2, r_1)$, etc. for arcs from places of type **CxR**.
- Arcs from a transition to a place (i.e., output arcs) should satisfy the same requirements unless they are conditional and of the form $[C]\%c$ or $[C]\%(c, r)$ (where C is some condition depending on the case attributes).
- Guards should be logical expressions created only using functions such as `match`, `has_role`, `has_orgunit`, etc. Function `match` can be used to make sure that all arc inscriptions involved in one transition bind to the same case id. Function `has_role` can be used to make sure that the resource selected has a given role. Function `has_orgunit` can be used to make sure that the resource

selected is a member of a specific organizational unit. The definitions of these functions are straightforward but beyond the scope of this paper.

- Each transition should have a code region. This is executed when the transition occurs. In the context of CWNs, code regions are used to link the case attributes of the input tokens to the output tokens. Since arc expressions cannot manipulate the case content, i.e., only route cases, this is the only way of changing the attributes of a case.

As indicated above, guards should only be used to make sure that only tokens corresponding to the same case are merged and that the tasks are executed by the proper resources. A typical guard is: `match(c1, c2) andalso has_role(r1, "manager") andalso has_orgunit(r1, "loans")`. We do not put any requirements on the code regions other than that the conservation of cases and resources should be guaranteed. Note that tasks are executed by humans using applications. In a CWN model one can try to model these but it will always be an approximation. Typically it is impossible to make a complete/accurate model of some bank application or bank advisor. Therefore, we can only try to approximate them and mapping the CPN code regions to some implementation language (e.g., BPEL) will have to be partly manual either at implementation/configuration-time or at run-time.

In this paper, we will not give a formal definition of CWNs. Rather, we focus on the CWN for the AP system.

4.3 Workflow Model for AP

Figure 6 shows the CWN, which is derived from the RCPN of Figure 3. The CWN reflects the parts in the RCPN that should be orchestrated by the workflow system. As stated in W2, it should not contain actions which are not supported by the system. We therefore leave out transitions `Observer customer enquiry`, `Lookup customer information and credit information` and `Refuse`, and the places `Customer observed`, `Early refusal`, and `Ready for advising`. The rest of the RCPN has been mapped into the CWN form as shown in Figure 6.

It can be observed that the CPN shown in Figure 6 is a CWN as defined in Section 4.2. It is easy to see that all places are of type `Resource`, `Case` or `CxR`. If the resource places are removed from the model, a WF-net emerges, i.e. all nodes are on a path from start to end node. In this context it is also evident that the net has a start and end node and is indeed sound, i.e., all cases begin from the place `Start` and end up in one of the places in the `End state` fusion set. Also the guidelines regarding the arc inscriptions and guards of transitions representing tasks are satisfied. Note the code regions in Figure 6 linking `input` to `output` via an `action` part.

Apart from satisfying properties (W1)-(W3) mentioned in Section 4.1, as a natural result of being created later than the requirements model, the workflow model improves a number of things — the workflow model corrects some errors and shortcomings in the RCPN and improves the modelling of a number of aspects. As an example, the RCPN contained traces which are not possible in the real world. For example, a loan could be given a prior approval and then rejected. In the real world a loan can only be granted if a prior approval had been given. Therefore we have restricted the behavior in the workflow model to avoid such execution paths.

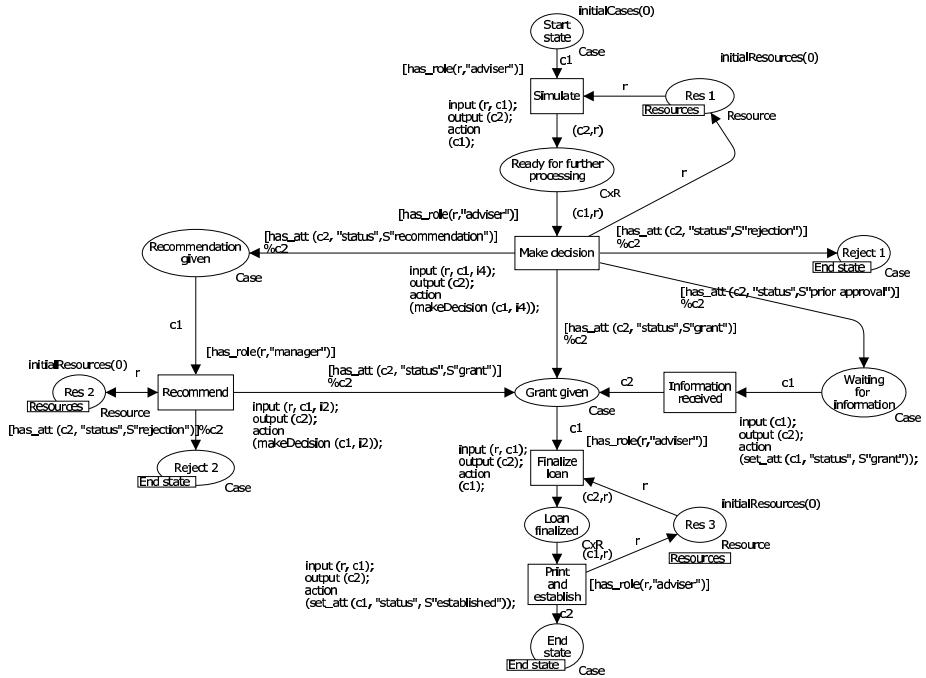


Fig. 6. The result of translation $T2$: the CWN for AP

In the following we give a structured translation of the requirements model to the workflow model beginning from the first activity in a blanc loan application process.

Generally, resource places are put in the model to reflect which human resources are available/needed at certain steps in the process. Note that place fusion [16] is used in Figure 6, i.e., the three resource places are actually the same place.

The first activity that occurs in the process is that an adviser proposes a loan setup in the module **Advising / Simulation**, this is mapped to transition **Simulate** which is the first possible activity which can occur in the workflow model. (Recall that the term “simulation” is part of the jargon used in banks and refers to a specific activity.) For this to happen an adviser must perform the work which is reflected by the arc from **Res 1** and the guard **[has_role(r, "adviser")]** on **Simulate**. The adviser resource is not put back before a decision about the loan is made in the transition **Make decision**. It is modeled in this way to reflect that it is the same adviser that proposed the loan which makes the decision.

The decision can have four possible outcomes as in the RCPN: (1) grant, (2) recommendation, (3) prior approval or (4) refusal. In contrast to the RCPN it is not possible to make this decision twice, i.e., the flow of the loan case will not loop back to the decision node **Make/review/change decision**. Below we describe how each of the four following scenarios are translated:

- In the RCPN, the case went into the module **Production** if a grant was given. This is reflected by the transitions **Finalize loan** and **Print and establish**. These are both done by the same adviser. This is reflected by the arcs connected to resource place **Res 3** and the guard of **Simulate**.
- If recommendation is given, the activity **Recommend** can occur. This is the translation of the module **Recommendation** in the requirement CPN. It is modeled by two outgoing arcs from the transition **Recommend** and a required manager resource from the resource place **Res 2**. In case of a refusal the case ends which is reflected by the case being moved to the end state of the workflow model.
- If a prior approval is given, the case moves to a waiting position on the place **Waiting for information** until the information arrives. When this happens, **Information received** occurs and the status of the loan is set to grant and the case follows the path for a loan with a grant from this point.
- When a refusal is issued, the blanc loan is moved to the end state with a rejection as status to reflect the termination of the case.

5 From Workflow Model to Implementation

Now we focus on translation *T2* of Figure 1. We do not present our technique to map a CWN onto BPEL in detail. For more details we refer to a technical report defining the mapping from WF-nets to BPEL [4].

The key issue is that CWNs can be used to semi-automatically generate BPEL code. It is possible to fully automatically generate template code. However, to come to a full implementation, programmers must manually complete the work, e.g., by providing the “glue” to existing applications and data structures.

BPEL offers many routing constructs. The atomic activity types are: **invoke** to invoke an operation of a web service described in WSDL, **receive** to wait for a message from an external source, **reply** to reply to an external source, **wait** to remain idle for some time, **assign** to copy data from one data container to another, **throw** to indicate errors in the execution, **terminate** to terminate the entire service instance, and **empty** to do nothing. To compose these atomic activity types, the following structured activities are provided: **sequence** for defining an execution order, **switch** for conditional routing, **while** for looping, **pick** for race conditions based on timing or external triggers, **flow** for parallel routing, and **scope** for grouping activities to be treated by the same fault-handler. Typically there are multiple ways of realizing the same behavior, e.g., the **flow** construct can be used to build sequences and switches but also the **sequence** and **switch** constructs may be used.

We have developed an iterative approach to “discover” patterns in CWNs that correspond to constructs in BPEL and generate template code based on this. The approach works as follows. First, the CWN is mapped onto an annotated WF-net. (Figure 7 shows an example of such mapping.) This involves removing the resource places and resource-related inscriptions, removing the color-related information, and replace transitions that represent choices by a small network. The resulting “uncolored” WF-net is annotated with information that can be used for the BPEL translation, e.g., conditions for choices. In the annotated WF-net we try to select a

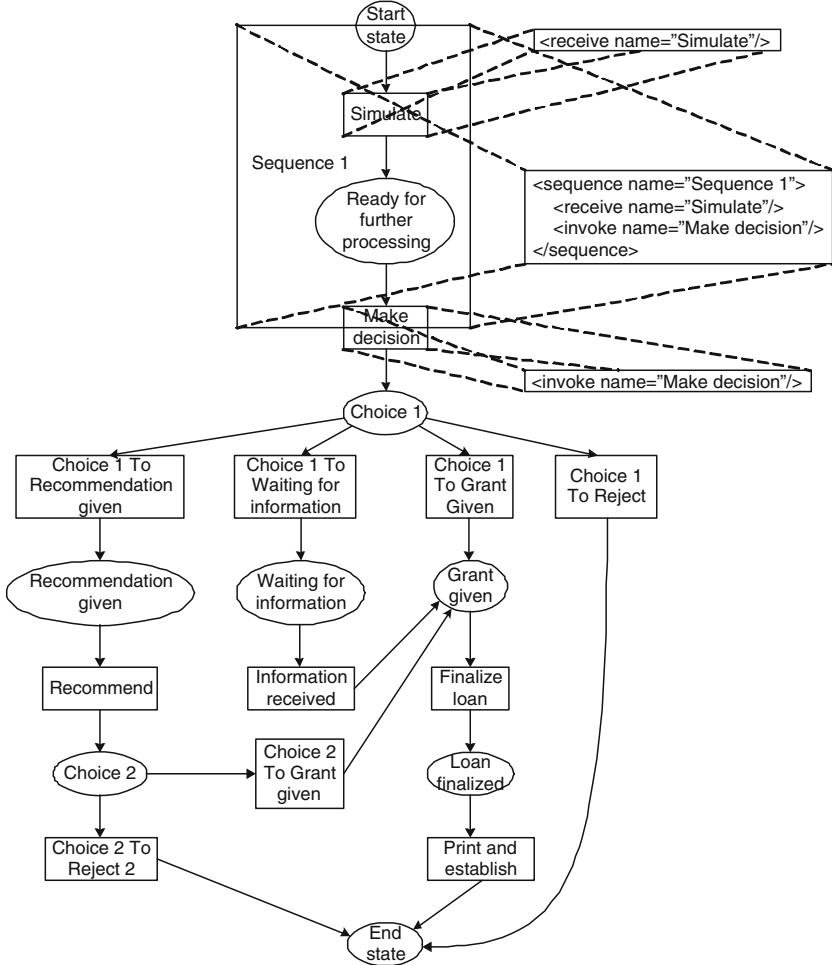


Fig. 7. The “uncolored” WF-net with some snippets of generated BPEL code

maximal sequence component, i.e., a sound workflow subnet that is both a state machine and a marked graph [1, 26]. This part of the net is reduced to a single transition representing the sequence (cf. Figure 7). If there is no maximal sequence component, the algorithm looks for some other structured component (e.g., a “switch”, “pick”, “while”, or “flow”) that is maximal in some sense. Again this component is reduced to a single transition. (The approach also allows for ad-hoc and reusable components with a corresponding BPEL mapping.) By applying these rules iteratively the Petri net is parsed completely and the parse tree is used to generate the BPEL code as illustrated by Figure 7.

We have applied the algorithm to map the CWN shown in Figure 6 onto BPEL code and tested this using IBM WebSphere Studio. The BPEL code and screenshots of WebSphere Studio can be downloaded from <http://www.daimi.au.dk/~krell/CoopIS05/>.

6 Related Work

Since the early nineties, workflow technology has matured [14] and several textbooks have been published, e.g., [2, 10]. Petri nets have been used for the modeling of workflows [2, 7, 10] but also the orchestration of web services [22]. The Workflow Management Coalition (WfMC) has tried to standardize workflow languages since 1994 but failed to do so. XPDL, the language proposed by the WfMC, has semantic problems [1] and is rarely used. In a way BPEL [5] succeeded in doing what the WfMC was aiming at. However, BPEL is really at the implementation level rather than the workflow modeling level or the requirements level (thus providing the motivation for this paper).

Several attempts have been made to capture the behavior of BPEL [5] in some formal way. Some advocate the use of finite state machines [13], others process algebras [12], and yet others abstract state machines [11] or Petri nets [23, 21, 24, 25]. (See [23] for a more detailed literature review.) For a detailed analysis of BPEL based on the workflow patterns [3] we refer to [28].

Most papers on BPEL focus on the technical aspects. This paper focuses on the life-cycle of getting from informal requirements to a concrete BPEL implementation based on a concrete case study. Therefore, the work is perhaps most related to the work of Juliane Dehnert who investigated the step from informal business models expressed in terms of Event-driven Process Chains (EPCs) to workflow models expressed in terms of WF-nets [8, 9]. However, her work is mainly at theoretical level and does not include concrete case studies or mappings onto some implementation language.

The work reported in this paper is also related to the various tools and mappings used to generate BPEL code being developed in industry. Tools such as the IBM WebSphere Choreographer and the Oracle BPEL Process Manager offer a graphical notation for BPEL. However, this notation directly reflects the code and there is no intelligent mapping as shown in this paper. This implies that users have to think in terms of BPEL constructs (e.g., blocks, syntactical restrictions on links, etc.). More related is the work of Steven White that discusses the mapping of BPMN onto BPEL [27] and the work by Jana Koehler and Rainer Hauser on removing loops in the context of BPEL [19]. Our work differs from these publications in the following way: we address the whole life-cycle (i.e., not a specific step or a specific problem as in [19]), we provide algorithms to support the mapping (unlike, e.g., [27]), and we use CPNs as a basis (i.e., start from a language with formal semantics).

The translation from WF-nets to BPEL (translation $T2$) is described in more detail in a technical report [4].

7 Conclusions

In this paper, we have used a real-life example (the new AP system of Bankdata) to go from a requirements model to a proof-of-concept BPEL implementation using the CPN language and CPN Tools. Figure 1 summarizes our approach and shows the different models and translations proposed. The focus of this paper has been on

translations $T1$ and $T2$. Essential for this approach is the use of the CPN language, first in unrestricted form (the RCPN) and then in restricted form (the CWN). The restrictions facilitate the automatic generation of (template) code.

In this paper, we introduced the CWN model and the translation to BPEL code. We believe that our approach can be generalized to other systems within and outside Bankdata. We also believe that our approach can be modified for other target implementation languages (e.g., languages used by systems of TIBCO/Staffware, FLOWer, COSA, and FileNet). Further case studies are needed to prove this point. We also aim at concrete tool support for the translation of CWN to BPEL. At this point in time, we provide only a manual procedure to accomplish this. We plan to develop a dedicated tool for this.

Another direction for future research is to develop techniques, tools, and animations specific for CWN. The CWN model can be seen as the high-level variant of the classical workflow nets, adding the data and resource perspectives. For workflow nets there are strong theoretical results, dedicated editors, and analysis tools [1, 26]. The goal is to offer similar support for CWNs. For example, it is interesting to explore different notions of soundness including the data and resource perspectives [1, 8, 15].

Acknowledgements. We thank Bankdata for allowing us to participate in the AP project. We thank the users and analysts we have worked with, in particular Gert Schmidt Sofussen, who has contributed significantly to the RCPN.

References

1. W.M.P. van der Aalst. Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 1–65. Springer-Verlag, Berlin, 2004.
2. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
3. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
4. W.M.P. van der Aalst and K.B. Lassen. Translating Workflow Nets to BPEL4WS. BETA Working Paper Series, Eindhoven University of Technology, Eindhoven, 2005.
5. T. Andrews, F. Curbera, et al. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.
6. C. Bossen and J.B. Jørgensen. Context-descriptive Prototypes and Their Application to Medicine Administration. In *Proc. of Designing Interactive Systems DIS 2004*, pages 297–306, Cambridge, Massachusetts, 2004. ACM.
7. P. Chrzastowski-Wachtel. A Top-down Petri Net Based Approach for Dynamic Workflow Modeling. In *International Conference on Business Process Management (BPM 2003)*, volume 2678 of *Lecture Notes in Computer Science*, pages 336–353. Springer-Verlag, Berlin, 2003.
8. J. Dehnert. *A Methodology for Workflow Modeling: From Business Process Modeling Towards Sound Workflow Specification*. PhD thesis, TU Berlin, Berlin, Germany, 2003.

9. J. Dehnert and W.M.P. van der Aalst. Bridging the Gap Between Business Models and Workflow Specifications. *International Journal of Cooperative Information Systems*, 13(3):289–332, 2004.
10. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems*. Wiley & Sons, 2005.
11. D. Fahland and W. Reisig. ASM-based semantics for BPEL: The negative control flow. In *Proc. 12th International Workshop on Abstract State Machines*, pages 131–151, Paris, France, March 2005.
12. A. Ferrara. Web services: A process algebra approach. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 242–251, New York, NY, USA, 2004. ACM Press.
13. J.A. Fisteus, L.S. Fernández, and C.D. Kloos. Formal verification of BPEL4WS business collaborations. In *Proceedings of the 5th International Conference on Electronic Commerce and Web Technologies (EC-Web '04)*, volume 3182 of *Lecture Notes in Computer Science*, pages 79–94, Zaragoza, Spain, August 2004. Springer-Verlag, Berlin.
14. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.
15. K. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 335–354. Springer-Verlag, Berlin, 2003.
16. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1997.
17. J.B. Jørgensen and C. Bossen. Executable Use Cases: Requirements for a Pervasive Health Care System. *IEEE Software*, 21(2):34–41, 2004.
18. J.B. Jørgensen and K.B. Lassen. Aligning Work Processes and the Adviser Portal Bank System. In *International Workshop on Requirements Engineering for Business Need and IT Alignment*, 2005.
19. J. Koehler and R. Hauser. Untangling Unstructured Cyclic Flows A Solution Based on Continuations. In *CoopIS 2004*, volume 3290 of *Lecture Notes in Computer Science*, pages 121–138, 2004.
20. L.M. Kristensen, S. Christensen, and K. Jensen. The Practitioner’s Guide to Coloured Petri Nets. *International Journal on Software Tools for Technology Transfer*, 2(2):98–132, 1998.
21. A. Martens. Analyzing Web Service Based Business Processes. In *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering (FASE 2005)*, volume 3442 of *Lecture Notes in Computer Science*, pages 19–33. Springer-Verlag, Berlin, 2005.
22. M. Mecella, F. Parisi-Presicce, and B. Pernici. Modeling E-service Orchestration through Petri Nets. In *Proceedings of the Third International Workshop on Technologies for E-Services*, volume 2644 of *Lecture Notes in Computer Science*, pages 38–47. Springer-Verlag, Berlin, 2002.
23. C. Ouyang, W.M.P. van der Aalst, S. Breutel, M. Dumas, A.H.M. ter Hofstede, and H.M.W. Verbeek. Formal Semantics and Analysis of Control Flow in WS-BPEL. BPM Center Report BPM-05-13, BPMcenter.org, 2005.
24. C. Stahl. Transformation von BPEL4WS in Petrinetze (In German). Master’s thesis, Humboldt University, Berlin, Germany, 2004.

25. H.M.W. Verbeek and W.M.P. van der Aalst. Analyzing BPEL Processes using Petri Nets. In *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 59–78. Florida International University, Miami, Florida, USA, 2005.
26. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
27. S. White. Using BPMN to Model a BPEL Process. *BPTrends*, 3(3):1–18, 2005.
28. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *22nd International Conference on Conceptual Modeling (ER 2003)*, volume 2813 of *Lecture Notes in Computer Science*, pages 200–215. Springer-Verlag, Berlin, 2003.

A Service-Oriented Workflow Language for Robust Interacting Applications

Surya Nepal^{1,*}, Alan Fekete², Paul Greenfield¹, Julian Jang¹,
Dean Kuo^{3,**}, and Tony Shi¹

¹ ICT Centre, PO Box 76, Epping NSW 1710, Australia

{Surya.Nepal, Paul.Greenfield, Julian.Jang, Tony.Shi}@csiro.au

² School of Information Technologies, University of Sydney, NSW 2006, Australia

fekete@it.usyd.edu.au

³ School of Computer Science, The University of Manchester, Manchester M13 9PL, UK

dkuo@cs.man.ac.uk

Abstract. In a service-oriented world, a long-running business process can be implemented as a set of stateful services that represent the individual but coordinated steps that make up the overall business activity. These service-based business processes can then be combined to form loosely-coupled distributed applications where the participants interact by calling on each other's services. A key concern is to ensure that these interacting service-based processes work correctly in all cases, including maintaining consistency of both their stored data and the status of the joint activities. We propose a new model and notation for expressing such business processes which helps the designer avoid many common sources of errors, including inconsistency. Unlike most existing orchestration or workflow^a languages used for expressing business processes, we do not separate the normal case from exceptional activity, nor do we treat exceptional activity as a form of failure that requires compensation. Our model has been demonstrated by developing prototype systems.

1 Introduction

The Web Services vision of next generation large-scale cross-organizational distributed applications will further drive both inter- and intra-business integration. Services are loosely coupled and the detailed implementation of a service is opaque to other interacting services, so the implementation of service can be changed without affecting other interacting services. One common way to implement large stateful server applications is to use a workflow language to combine and orchestrate a number of individual services such as checking the customer's status, checking stock levels, asking for and receiving payments, checking that the invoice is fully paid, or sending notification of a delivery date in an ecommerce application.

The major vendors are proposing Web Services standards for defining business processes, including BPEL4WS [1] and WSCI [2]. These proposals use existing ideas

* This work is completed as part of CeNTIE project that is supported by the Australian Government through the Advanced Networks Program of the Department of Communications, Information Technology and the Arts.

** This work was completed while the author was working at CSIRO.

from the database community, including workflow descriptions based on graph-like arrangements of the steps, and advanced transaction models with compensators and failure-handlers for nested scopes. We have previously discussed the shortcomings of this approach for programming consistency-preserving services in [14].

This paper offers an alternate way to express the workflow within such a business process. Our approach, which we call GAT (standing for guard-activity-triggers), extends previous work on event-based programming models and adds several features that help the software designer avoid common classes of errors.

In our GAT model, a workflow involves the coordinated execution of a number of individual stateful actions. Each of these actions is just a conventional code segment, perhaps a legacy transaction, that may access and modify persistent data. Unlike the proposed graph-based standards, in the GAT model the flow of control between actions is not defined explicitly. Our approach instead is based on the event-driven model. Actions are invoked when a corresponding event is raised, and only if appropriate guard condition holds. This programming style is often called ECA, and is common in active databases and agent-based systems [28], and has been proposed for workflows [6, 7, 25].

Our GAT proposal goes beyond previous ECA workflow languages and has a number of key features which together assist a software developer in avoiding a number of common mistakes.

- **Uniform processing:** there is no separation of the “normal” case from “exceptional” processing. All events are equal and just result in the execution of their corresponding actions.
- **Resumption:** business processes can execute actions handling “normal” cases after executing actions handling ‘exceptional’ cases.
- **Access to state:** there is no hidden or implicit state. The equivalent of the implicit ‘current position’ state in a graph-based workflow is replaced with normal state that can be examined and updated from within actions.
- **Uniform outcome:** for the business process as a whole, there is no inbuilt notion of return to an initial state or compensation. Individual actions may act as atomic transactions and abort and rollback, but the whole GAT process merely continues executing actions as they become enabled until the process completes. Of course, the software designer may choose to define actions that act as compensators for others.
- **Coverage:** alternate actions for the same event are grouped together and conditions specify which of these actions should be executed. Simple “closure” tests on conditions can guarantee that at least one action will be executed whenever any event is raised.
- **Protected action:** each action has a guard that is sufficient to ensure the code of the action runs without errors. This prevents common mistakes arising from lack of isolation where the code within an action assumes some property (such as existence of the customer) that was checked earlier in the process, but may have since been modified by concurrent processes.
- **Response to non-occurrence:** the trigger mechanism used to raise events that drive business processes forward can also be used to detect the expiry of a timeout, so actions can be taken when expected messages fail to arrive.

- **Integration:** The raising of new events is separated from the action code that modifies state, allowing legacy code to be used as actions. This adds to the ECA model the same uncoupling of control flow and processing steps that is found in graph-based models.

The significant contribution of the GAT model is in the way that it combines these features to assist developers in the construction of more robust distributed applications.

One of the most important robustness properties of a business process is that the application will always terminate in one of a specified set of consistent “acceptable” states. Some of these acceptable states may correspond to “successful” outcomes, such as goods received and paid for in full, while others represent “less desirable” outcomes, such as purchase order cancelled and cancellation fee paid. These consistent outcomes are all acceptable to all the participants in the distributed application, and none of them are treated as “failures” or “successes”. This is supported by the “uniform outcome” feature of the GAT model.

Another key robustness property of a business process is the complete specification of an application. The application must be able to handle all arriving messages at any stage of its execution. Some of the messages may arrive even after they are no longer expected by the application. For example, the merchant may receive the cancellation request from the customer when the goods are in transit or delivered. This is supported by the “coverage” and “access to state” features of the GAT model. The application must also be able to deal with non-arrival of an expected message such as when the merchant has not received the payment even after the due date. This is supported by the “response to non-occurrence” feature of the GAT model.

The safe and concurrent execution of services is another important requirement for a robust distributed application. For example, the merchant must be able to run payment service concurrently with delivery service without violating business rules, defined by integrity constraints on application data. The “protected action” feature of the GAT model supports this requirement.

The rest of the paper is organized as follows: section 2 describes an e-procurement application as well as discussing the problems that need to be solved. Details of the GAT approach are given in section 3. We carried out an extensive case study for an e-procurement system and this is discussed in section 4. We have used the GAT approach to develop an executing example system based on .NET technology and in section 5 we show how a designer can build applications based on the GAT concepts. Related work is presented in section 6.

2 A Motivating Situation

Fig. 1 shows an e-procurement example. It includes a customer, a merchant, (two) shippers and (three) suppliers. The customer initiates the business process by sending a quote request. When the merchant receives a quote request, it responds with a quote or a rejection message. If the customer service decides to go ahead with the purchase, it will send a *purchase order* message and the merchant service will then confirm the order by sending a *confirmation* message. Messages for coordinating payment and delivery such as *invoice*, *payment*, *receipt*, *goods shipped* and *goods received* are

normally sent and received after the *confirmation* message. The merchant service also exchanges messages with suppliers to order goods if they are not available in the warehouse. Messages for coordinating supply of ordered goods and delivering them to the warehouse by a shipper are then exchanged between the merchant service, supplier services and shipper services.

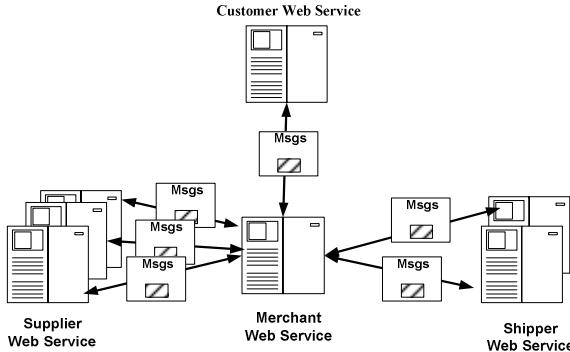


Fig. 1. E-Procurement Scenario

The purchase process is completed when the customer pays for the invoice (i.e., receives the receipt) and receives the goods (i.e., sends an acknowledgement message for the goods received); similarly, the merchant delivers the goods and receives the full payment.

2.1 Problems

We described the behaviours of services in a purchase process when everything goes according to plan above. Such behaviours can be specified easily with business process modelling languages such as BPEL4WS and WSCI. These approaches have shortcomings in describing robust applications. For example, they use fault and compensation handlers to deal with exceptional cases. Describing all ways of processing exceptions using fault and compensation handlers is either clumsy or impractical due to the interactions between each handling and the rest of the business process [14], and the developer may easily introduce errors.

A business process must seek to avoid common errors, including inconsistency. The consistency requirements are business process dependent. For example, the merchant service is consistent at termination when it receives the full payment and delivers the goods; whereas the customer service is consistent when it receives the goods and pays for the invoice. We next describe a few examples that are difficult to express in standard approach and could be sources of inconsistency.

Threats to consistency can arise due to the lack of **coverage** if the merchant's code doesn't fix all the issues when a customer's cancellation is received after shipment has been arranged, leading to an outcome where the customer is unwilling to receive goods which are in transit! Moreover, if the customer's cancellation request is rejected, rather than treating it as a fault and rolling back the entire sale process, the merchant must **resume** the normal processing. The importance of offering resumption in dealing with failures was identified by Goodenough [12].

An example of failure due to **unprotected action** is if the code for issuing an invoice fails catastrophically when a concurrent activity has changed the customer's status in the database after an earlier step checked that the status was appropriate. Similarly, rather than using compensation to back out from the entire sale, the merchant must seek an alternate payment method if the selected method could not be used for some reasons.

Incomplete specification is one of the common mistakes programmers make while describing business processes. Every possible message must be handled and there must be some way to proceed if an expected message does not arrive. Asynchronous communication between services may make the processing a message even more difficult. The message may come at any time. For example, the merchant may receive a payment while processing a previously received payment. The execution of a correct action depends not only on the arrival of a message, but also on **access to states**. For example, if a payment is received, the correct action to be executed depends on whether this completes the amount due, or leads to an overpayment, or leaves some amount still owing. Moreover, if a process is expecting a payment and the payment has not been received by a due date, the process must send a reminder as a **response to non-occurrence** of a payment.

The distributed application must be able to handle these situations and make sure that all participants in the distributed application always maintain the consistency. The challenge to application programmers or developers is to write the individual services participating in a distributed application in such a way that they maintain consistency despite concurrency, exceptions and failures. We next describe the properties that a process must have to avoid the problems discussed in this section.

2.2 Robustness Properties

The fundamental robustness property is that a process must always terminate in an acceptable state even if there are concurrent processes executing, and regardless of the outcomes of any activities within process. In order to achieve this goal, the programming model should have the following features.

- Safe concurrent execution: Arbitrary number of activities from different processes may access and update share data. Activities must be able to execute concurrently without corrupting data; otherwise, the system may end up in an inconsistent state. This property guarantees an isolated execution of activities.
- Termination in acceptable distributed state: An activity may have more than one outcome. The process must be able to deal with all these outcomes and move towards the defined set of acceptable termination states. There could be more than one acceptable state as they are dependent on business rules and defined by application programmers. There is no such state as failure. Once the process has begun, the process always terminates in an acceptable termination state and the states are explicit.
- Distributed process always complete: There are no deadlocks, all participants should be able to complete and there are no messages left unprocessed at termination.

- Completeness of components: The process should be able to handle all messages received under all possible circumstances; that is, activities must be defined for all possible combinations of states and messages.

3 GAT Programming Model

This section discusses the GAT programming model and how it provides the features that are needed to build robust distributed applications.

A process is defined by a set of *activity groups* where each activity group consists of an *event* (a newly arriving message) and a set of related *activities*, as shown in Fig. 2. All arriving messages, whether they deal with normal or exceptional processing, are treated **uniformly**.

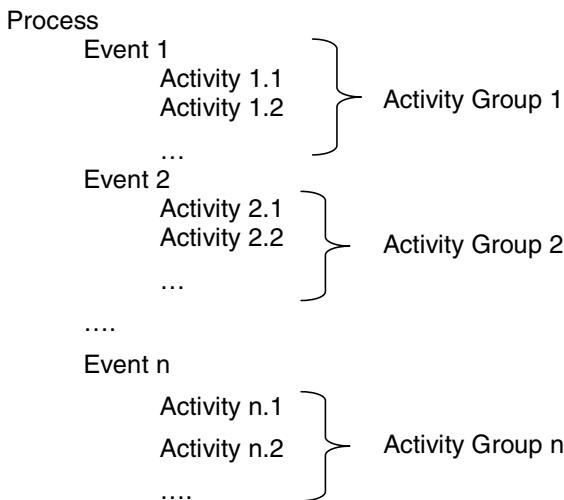


Fig. 2. Activity Groups and Activities

Fig. 3 shows just one activity from an activity group. Each activity represents one possible response to its event and consists of a *guard*, an *action* and a set of *trigger groups*. Each *trigger group* consists of a set of *triggers*. Guards are Boolean expressions that can refer to parameters that came with the event (message fields) and also they can **access the state** (referring to variables in the workflow engine), and the guard controls whether or not their corresponding action should execute as part of the response to the event. The action part of an activity is conventional code written to correctly handle the incoming event, this code is **protected** because it can assume the properties checked in the guard. Trigger expressions are evaluated after the action has completed to determine what other events need to be raised to further handling of the initial event. The separation of triggers from action enables developers to **integrate** legacy codes with GAT structure.

Fig.3 shows an activity composed of a guard, an action and two trigger groups, the first containing two triggers and the second containing four triggers.

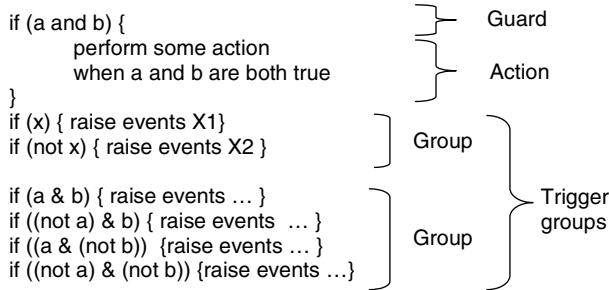


Fig. 3. Structure of an activity

A trigger consists of a condition and a set of events (messages). Each trigger condition expression is evaluated in turn and the corresponding events are sent if the condition is true. These events can be sent immediately or can be *deferred* for a specified period of time. The deferred events is used to compose a **response to non-occurrence** of an expected message.

The guard expressions in any one activity group are *closed*, meaning that the guard of exactly one activity in an activity group has to be true. That is, in an activity group, when an event is received, the Boolean expression of one of the activities *must* be true and all the other Boolean expressions *must* be false. This property lets us guarantee **coverage**, so that we will always take some action every time an event is received. In order to meet closure requirements, this group must also include other activities that will specify what should be done if the expression (*a and b*) does not turn out to be true.

The trigger expressions in each trigger group are also *closed*. That is, in a single trigger group, exactly one trigger expression must be true and only events in that trigger expression will be raised as a result. Activities can have multiple independent trigger groups, each corresponding to a different and parallel possible course of action.

Fig. 3 shows an activity with two trigger groups, both of which are closed. This property lets us guarantee that we will always consider all possible outcomes of an activity. The formal model is omitted here due to the limitation of space (see [24]).

In the GAT model, evaluation of guards and the execution of action as well as evaluation of trigger conditions and raising events form an isolated unit (under locks that prevent interleaving by other concurrent actions) in order to ensure the action is properly **protected**. Note however that response to any events raised due to the triggers will be in a different isolation block.

In other work [19], we have proposed a related model for the external interface (choreography) of a service, where again there is no separation of success from failure outcomes.

4 Case Study: Payment Process

The aim of this section is to illustrate how the GAT programming model has addressed the problems identified in section 2.1 above. The overall structure of the process that implements the normal behaviour of the merchant service is shown in

Fig. 4. The normal payment process consists of sending an invoice, receiving payment and sending a receipt. While we concentrate on payment (sub)process in the rest of this section, similar issues arise in other (sub)processes within the merchant process such as quote, purchase and delivery.

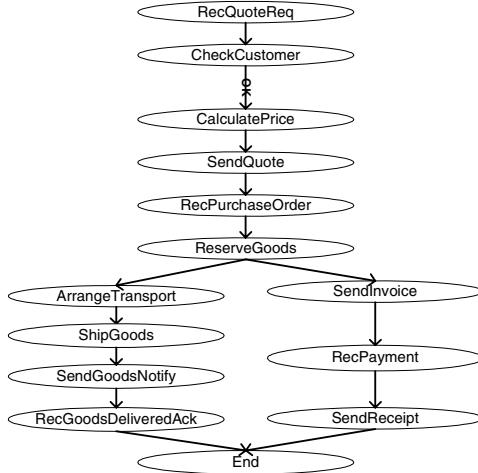


Fig. 4. Merchant's Service

There are a number of other possible execution paths in the payment process not shown in the figure. For example,

- If the payment does not arrive by the due date, the merchant sends a reminder and extends the due date. If the merchant receives the payment within the extended due date, the merchant sends the receipt and terminates the payment process.
- If the merchant receives an order cancellation request from the customer before receiving a payment, the merchant may accept the cancellation request and terminate the payment process.
- The merchant rejects the cancellation request and proceeds ahead if it receives the cancellation request when the goods are in transit.

There exist other possible execution paths due to race conditions such as if the full payment was in transit when the merchant sends non-payment notification message to the manager or if the late fee notification was in transit when the customer sends the full payment. There exist another large number of possible execution paths in other sub-processes. Representing all these possibilities in graph form as shown in Fig. 4 is very clumsy, and soon becomes infeasible for practicing developers.

The payment process has four acceptable termination states: cancelled, paid in full, non-payment and refund. It does not have any failure state as indicated in Section 2.3. The above description of payment process clearly indicates that even when an activity within the process fails or an exception occurs, the process terminates in one of the acceptable termination states.

We next describe merchant's payment process in GAT programming model and illustrate how it addresses the problems identified in Section 2.1. It also illustrates how easier it is to construct business processes that cover a large number of complex cases using just a couple of activity groups.

Send Invoice

The activity group "sendInvoice" prepares and sends an invoice to the customer. Its event is raised when the ordered goods have been reserved successfully. The activity group contains two activities, the first executes when the customer is valid (e.g., there is no overdue payment for the customer) and the second activity executes when the customer is invalid.

The action part of the first activity in the group prepares the invoice and the triggers sends it. The guard **protect the action**, so that the customer remains valid during its execution. It also sets a *deferred* event to receive the payment from the customer within due date. That is, the event "overduePayment" is set, but not fired; it is fired if the condition "not(PaidinFull)" is true even after the "duedate".

The action part of the second activity in the group prepares a notification message and the triggers sends it to the customer. If the customer pays the overdue payment within due date, the process **resumes** as normal. Otherwise, the merchant raises an event "invalidCust" and defines appropriate activity groups to deal with it (not defined here). This also shows how GAT provides **uniform processing** methods for both valid and invalid customers.

Group: sendInvoice	
Event	goodsReserved
Activity: invoicing	
Guard	<i>The customer is valid</i>
Action	<i>prepare invoice message invoice to be sent to the customer;</i> <i>set PaidInFull to be False;</i> <i>set balance to be the invoiced amount;</i> <i>set duedate for the payment from the customer to 30 days;</i> <i>construct the overduePayment message to be sent back to its own process if the full payment is not received by the due date ;</i>
Triggers	(True) \Rightarrow invoice (True) \Rightarrow (not(paidinFull), duedate, overduePayment)
Activity: cancelling order	
Guard	<i>The customer is invalid</i>
Action	<i>construct a notification message invalid to be sent to the customer;</i> <i>set duedate for the response from the customer to 7 days;</i> <i>construct the invalidCust message to be sent back to its own process if the customer is still invalid by the due date ;</i>
Triggers	(True) \Rightarrow invalid (True) \Rightarrow (not(custValid), duedate, invalidCust)

Process Overdue Payment

Distinctive processing (corresponding to an exception in other models) may occur due to timeouts, in **response to non-occurrence** of messages. Timeouts are a critical part of most business processes. Whenever a process is waiting for an incoming message, there is usually some limit placed on how long it can wait and the path of execution may change when a wait times out.

If a full payment is not received within due date, an exception caused by timeout occurs. In this situation, a reminder is sent to the customer. If the payment is not received fully in response to this reminder, then the reminder is re-sent. If the payment is not received fully after three reminders have sent out, then an alarm notification is raised to the manager. Mourani and Antunes [22] describe a way that allows user involvement in workflow exception handling. This is also possible in our model by raising appropriate events in triggers that report to human such as “alarmMsg” in our example below.

Group: overduePayment	
Event	overduePayment
Activity: sendReminder	
Guard	<i>Full payment has not been received from the customer and fewer than 3 reminders sent</i>
Action	<i>Extend the duedate by 7 days; increment the sent reminder counter; construct the reminderNotification message; construct the overduePayment message to be sent back to its own process if the full payment is not received by the due date ;</i>
Triggers	(true) \Rightarrow reminderNotification (true) \Rightarrow (not(paidInFull), duedate, overduePayment)
Activity: notPaidInFull	
Guard	<i>Full payment has not been received after sending three reminders</i>
Action	<i>construct an alarm message alarmMsg to be sent to manager set non-payment to True;</i>
Triggers	(true) \Rightarrow alarmMsg
Activity: receivedFullPayment	
Guard	<i>Full payment has been already received // no need to do anything</i>
Action	<i>// do nothing</i>
Triggers	<i>// do nothing</i>

Process Payment

The customer responds to an invoice by sending back payment. The customer may choose two options for payment: full at once or instalments. The “recPayment” activity group is illustrated in GAT syntax in Section 5. The activity groups “sendRefund” and “sendReceipt” are not shown here due to the limitation of space (see [24]).

Process Cancellation

Payment may be cancelled due to the cancellation of the order. Cancellation may be requested at anytime during the payment process. However, cancellation need not mean arriving in a state equivalent to having no order at all, since (depending on the business logic in the ordering processing) there may be a cancellation fee.

The simplest cancellation scenario is where the cancellation request is received anytime before the payment is received, the payment process can be terminated without taking any further action if the cancellation does not attract any cancellation fee. Note that some actions might be taken in the larger ordering process such as cancellation of reserved goods and cancellation of booked shipper. However, our discussion here shows only actions in the payment process. A slightly more complex class of cancellation is the one where the cancellation request is received after the payment is received, but before the goods in transit. The cancellation of the process then only requires the refund of the received amount or the remaining amount after deducting the cancellation fee. If the cancellation request is received when the goods are in transit, the merchant rejects the cancellation request and continues the normal processing, that is, we must have **resumption**.

The activity group that deals with the cancellation request in the payment process is defined below. The right behavior of the payment process due to cancellation depends on **access to state** of the concurrently running processes in the merchant, such as the “goods in transit” property in the delivery process. There are three activities in the activity group to handle these three different scenarios.

Group: recCancelRequest	
Event	cancelRequest
Activity: paymentNotReceived	
Guard	<i>Payment not received and goods not in transit</i>
Action	<pre> set cancelled to be True; set cancelfee to be True if the cancellation occurs after 7 days of order confirmation; if cancelfee { construct a message cancellationInvoice to be sent to the customer; } construct an event cancelConfirm to be sent to the customer;</pre>
Triggers	$(\text{true}) \Rightarrow \text{cancelConfirm}$ $(\text{cancelfee}) \Rightarrow \text{cancellationInvoice}$ $\text{not}(\text{cancelfee}) \Rightarrow // \text{ do nothing}$
Activity: paymentReceived	
Guard	<i>Payment received and goods not in transit</i>
Action	<pre> set cancelled to be True; set overpaid to be True if the received amount is greater than cancellation fee; if (overpaid){ construct an event overPayment to be sent back to its own process; construct an event cancelConfirm to be sent to the customer;</pre>
Triggers	$(\text{true}) \Rightarrow \text{cancelConfirm}$
Triggers	$(\text{overpaid}) \Rightarrow \text{overPayment}$ $\text{not}(\text{overpaid}) \Rightarrow // \text{ do nothing}$
Activity: goodsInTransit	
Guard	<i>Goods in transit (payment received or not)</i>
Action	<i>construct a cancellation rejection message rejectCancellation to be sent to the customer;</i>
Triggers	$(\text{true}) \Rightarrow \text{rejectCancellation}$

We ensure **coverage** by the GAT model requirement that the guard conditions on three activities are closed; that is, only one of the guards is going to be true when the merchant receives the cancellation request. The first and second activities have two trigger groups that are closed; that is, exactly one trigger condition from each group is going to be true. The closure properties on guards and triggers thus forced developers to define complete specification handling cancellation request within payment sub-process.

The rejection of cancellation request is not well handled by the standard fault/compensation approach. If a customer's cancellation request is rejected, then normal processing should continue. The most natural method to handle cancellation request in the standard approach is to throw a fault, but there is no way to handle a fault and then go back to normal processing, thus it is difficult, if not impossible, to implement the required behaviour. However, this is easy to implement in GAT model as shown above by the third activity in the activity group "recCancelRequest".

5 Prototype Implementation

Fig. 5 depicts the overall design for the merchant service as a specific example to illustrate different components of the prototype system implementation and the technologies, which are used to support each aspect. We next describe important components.

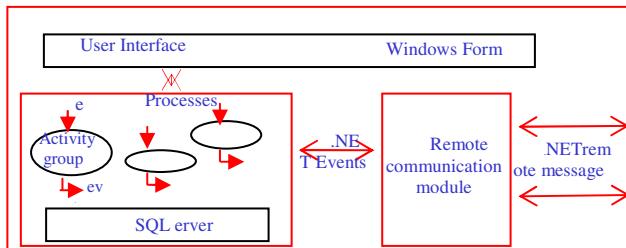


Fig. 5. Design for the merchant service

User Interface: User interface is a front end presentation layer to capture data from human operators (i.e., purchase order form, payment form etc.,) or to render a collection of data received from the processes. The service uses this layer to monitor the business activities such as what messages have been received from (or sent to) other services.

Process: Process is a core part of the implementation where business logic is defined. The designer writes each process with a structure corresponding to a set of activity groups and activities using GAT syntax. Below is an example merchant "recPayment" activity group in GAT syntax.

```

IN_EVENT: Payment // incoming event
ACTIVITY GROUP: ProcessPayment
  ACTIVITY: ProcessFullPayment
    GUARD: FullPayment(payment) <AND> sent(Invoice)
    ACTION: UpdateInvoice(payment)
    TRIGGERS: {true} <INT> PaidInFull(payment)
  
```

```

ACTIVITY: processUnderPayment
GUARD: UnderPayment(payment) <AND> sent(Invoice)
ACTION: UpdateInvoice(payment)
TRIGGERS: none
ACTIVITY: processOverPayment
GUARD: OverPayment(payment) <AND> sent(Invoice)
ACTION: UpdateInvoice(payment)
TRIGGERS: {true} <INT>OverPayment(payment)
ACTIVITY: processFaultPayment
GUARD: <NOT> sent(Invoice)
ACTION: ProcessFaultPayment(payment)
TRIGGERS: {true} <OUT>CustomerRemoteService.FaultPayment(payment)

```

The prototype implementation has a translator that converts the GAT specification to running code. We use C# as an underlying language for program codes; however, the .NET framework would allow us to work with different programming languages.

Events: The engine generates new events and raises (publishes) them as necessary with event data each event can carry. Activity groups subscribe to specific events so that an activity group can be notified about an event when it is raised. For example, in the merchant service, the activity group ‘ProcessPayment’ subscribes to the event ‘Payment’. When the customer sends in a payment, the engine raises an event ‘Payment’ that is to be signaled to the merchant’s activity group ‘ProcessPayment’. There are four types of events: incoming, outgoing, internal and conditional.

We use three interrelated elements of .NET Event mechanism to define internal events: a class that defines events and event data, then raises (publishes) events, and these events are consumed by interested event subscribers. The code below shows how to convert GAT event to .NET Events.

```

// defining an event and an event data
public static event PaymentEventHandler ePayment;
public class PaymentEventArgs : EventArgs {....}
// raising an event
public virtual void OnEPayment(PaymentEventArgs paymentArgs) {....}
// consuming an event
this.ePayment += PaymentEventHandler(ReceivePayment)
Public void ReceivePayment(object sender, PaymentEventArgs pArgs) {...}

```

An additional step is required to implement outgoing and incoming events. A proxy object needs to be created first to communicate with the remote service. Then the remote methods can be called in the same way as methods exist in a local service. The remote communication module listens for any incoming and outgoing messages. The module converts the incoming .NET remoting messages to internal events and sends them to the activity groups. Similarly, the module receives certain outgoing events raised by the activities, and converts them to outgoing .NET remoting messages to other services.

A different approach is used to implement a conditional (or deferred) event, as it is invoked immediately but consumed after specific period of time. We use an intermediate event handler that consumes a direct event invocation. At the same time it starts a timer event that points to another method. Once the timer expires, the other method will be invoked. The overall effect is the event is fired immediately, but actually consumed later.

Inter Process Communication: The incoming and outgoing messages are used for inter process communication and they are asynchronous in nature. These messages are handled by remote communication module. The module supports asynchronous programming in .NET remoting scenario. Developers define objects with methods to receive remote calls. Then they wrap the objects with the specific object (AsyncDelegate) to indicate that the methods are to be called asynchronously. Below example shows sending asynchronous quote requests using remoting.

```
//defining a callback method
public void OurRemoteAsyncCallBack (IAsyncResult ar) {..}
//sending asynchronous quote requests
public void SendQuoteRequest(object sender, QuoteRequestEventArgs e){..
    AsyncCallback RemoteCallback =
    new AsyncCallback(this.OurRemoteAsyncCallBack);
    RemoteAsyncResult RemoteAr=
    new RemoteAsyncResult(ms.ReceiveQuoteRequest );
    IAsyncResult RemAr=
    RemoteDel.BeginInvoke (e.quoteRequest,RemoteCallback,null ...}
```

Data storage and access: Any mission critical data to run the business activities are kept in a persistent storage. The data is accessed by the code representing activities, and manipulated for further processing. For example, the merchant service saves the payment information in a SQL database, and accesses this during guard evaluation. The activity group uses ADO.NET classes to connect to these data sources, and retrieve and update data.

Activity group: In the running code, an activity group is coded as a functional method within an outmost {}, and each activity is defined inside the if-else-then block whose exclusive business case is defined at an “if clause”. The guard is a conditional expression evaluated in an if statement. An action is a piece of code written in C# (or any conventional languages such as C++ or Java) to carry on business logic in a circumstance evaluated by the guard expression. The following code is generated for the merchant’s payment activity group described above in the GAT syntax.

```
// Activity Group ProcessPayment
public virtual void ProcessPayment(object sender, PaymentEvent e) {
    // Activity : processFullPayment
    if ((mp.FullPayment(e.payment))) {
        // guard evaluation run action code generate trigger(s) }
    // Activity : processUnderPayment
    if ((mp.UnderPayment(e.payment))) { ....}
    // Activity : processOverPayment
    if ((mp.OverPayment(e.payment))) { ...}
    // Activity : processFaultPayment
    if ((mp.FaultPayment(e.payment))) { ...}}
```

We now show the sequence of steps involved in the execution of a merchant’s code for receiving a payment.

- The customer submits a payment via payment form. The customer’s remote communication module sends this as a “Payment” message to the merchant.

- The “Payment” message is received by the merchant’s remote communication module. The module converts and raises the “Payment” message as an event ‘e’ with an event object ‘payment’.
- The event ‘e.payment’ is then received by the activity group which subscribes to this event.
- When the event is received by the activity group, the guard expression is evaluated. Also the action of the activity whose guard expression evaluates true is executed. On completion of the action, trigger expression is handled and the event whose trigger condition evaluates true is executed.
- On the completion of the execution, the engine produces the log file which shows the list of incoming events and which activity group send/receive them with the timestamp.

We draw the following lessons from the prototype implementation. We are able to establish that it is possible to implement real service oriented distributed applications with GAT model using widely-accepted technologies. We are also able to demonstrate that it is possible to check correctness properties of the specification at compile time. Those correctness properties include that all events are handled, guard conditions are closed and trigger conditions are closed. The implementation also provides us an insight about the isolation requirements of the GAT model. We found that the level of isolation required depends on the types of state. This observation will lead to further work on classification of types of states. Our implementation also provides the following guidelines for the developers to build a robust distributed application. We also develop tools to check the compliance of these guidelines at compile time.

- All guards and triggers must be closed.
- At least one activity group must be defined for each incoming event.
- An activity group defined for an incoming event must have a navigational state in guard if the incoming event is expected a response to the outgoing event for which a conditional event is defined. For example, the “payment” activity group must have navigational state “sent(Invoice)” in guards.

6 Related Work

The database community has very well accepted ways to help application programmers ensure consistency and robustness in OLTP systems, through the notion of ACID transactions, supported by locking and logging [13]. However, since the early 1980s, much effort was given to exploring advanced transaction models that might offer similar assistance to developers of long-running activities, where holding locks throughout the activity is infeasible for reasons of performance or system autonomy. Many of these models can be used with business process or workflow systems, as shown by Alonso et al. [4]. We concentrate here on the approaches that have been important in the workflow community. By far the most influential advanced transaction model for workflows has been Sagas [9]. In this model, a long-running activity is constructed from a sequence of steps each of which is atomic, and so can be aborted if a failure occurs while the step is running. If this happens, or if the whole sequence is unable to complete for any reason, then the system will invoke user-supplied compen-

sators in reverse chronological order for each step that did commit. This is the model used in the proposed standards for Web Services, such as BPEL4WS.

Another advanced transaction model that had less uptake elsewhere, but has influenced our GAT model, is called ConTract [27]. One key aspect is that each step has an associated condition which must be checked dynamically just before the step takes place, to ensure that all accessed state is suitable for the step. These conditions were checked by previous steps in the workflow, but ConTract introduced the idea of repeating the check to protect against errors caused by concurrent processes which are not isolated. These conditions inspired the GAT models guards.

There are many sorts of failure or exception that can arise in workflows. These were catalogued by Eder and Liebhart [8]. Early workflow systems like FlowMark only aimed to deal automatically with system failures [3], but later models [20] have seen more types such as semantic failures and engine failures encompassed in the exception-handling framework.

There is a huge literature on notations for describing business processes, workflows, or long-running activities. Many of these notations have been implemented in systems offering workflow-management or business process support [10]. The dominant approach in commercial systems presents a graph controlling the flow of control between steps, or as a simplification of this, a block-structured language with fork and join constructs as well as sequential flow and conditional branching. This is also the model used in standards such as BPEL4WS [1], and WSCI [2]. A smaller group of research papers and prototypes however have considered an event-based approach similar to the one in our GAT model. For space reasons, we will here focus on the most closely relevant research, which is event-based.

The initial impetus for event-based control flow came from the flurry of research in active databases[28] where triggers are used to respond to situations whenever they occur. The first paper to adopt this idea for managing control flow in a long-running activity was Dayal et al. [7], where the ECA (Event-Condition-Action) notation was proposed. The idea has proved especially valuable for building prototypes of distributed workflow execution engines such as C²offein[20], IRules[25], EVE [25] and WIDE [6]. Semantics for the ECA model are proposed by Geppert et al. [11]. Key features of our GAT model not found in these ECA systems include the grouping of actions with closure property on the conditions (to ensure coverage), the capacity to raise events which will occur later but only if some appropriate condition does not happen in the meantime (in order to provide time-outs), the concept of final events, and the idea of uniform outcome.

One paper does suggest some grouping and coverage condition: Knolmayer et al. [18] have proposed an ECAA (Event-Condition-Action-AlternateAction) model, which is in essence an activity group of exactly two actions with conditions that are complementary to one another. The paper mentions the possibility of larger numbers of actions being grouped, but gives no details.

Several other proposals have used ECA rules for dealing with exceptional conditions, while graph models are used for normal case processing (these are often called the “blue sky” paths). These proposals focus especially on the need to adapt and vary the way exceptions are handled, as the system evolves. Casati et al. have defined a language (Chimera-Exc) and implemented a system FAR [5], Hagen and Alonso built

OPERA [15, 16, 17], and Muller et al describe AGENT WORK [23]. Because these systems do not offer uniform handling, and they terminate the normal case when the exception is raised, thus they have difficulties in all the situations we described above where resumption and access to state are needed to decide the proper response to a cancellation or other exception.

Most of the workflow literature deals with a single uniform workflow engine within which the entire business activity occurs. Our emphasis on interacting concurrent workflows, as in a service-oriented system, seems novel. The closest relevant work we know is [17], where an ECA style is used for interaction between subprocesses within a single workflow engine.

7 Conclusions

We have proposed a new model and notation for expressing interacting business processes. The proposed model goes beyond previous workflow languages. It has a number of key features which together help the designer avoid many common sources of errors, including inconsistent outcomes. Unlike most existing workflow languages used for expressing business processes, we do not separate the normal case from exceptional activity, nor do we treat exceptional events as failures that require compensation.

Defining a normal behavior in a process is easy and straightforward in the standard graph-based languages. But the same can not be said for the exceptions and failures. The main reason is that the standard approach uses fault handlers or compensation handlers to deal with most exceptional situations. Such approaches can handle certain class of exceptions, but not all possible exceptions. Moreover, handling exceptions leads to the termination of a process. Describing all possible execution paths including exceptions using just fault and compensation handlers becomes either clumsy or impractical.

The event-based GAT model presented in this paper overcomes the limitation of existing programming models by effectively getting rid of exceptions. That is, there is no distinction between the exception and normal processing. Guards always define the correct action to take when an event occurs, taking into account of the current system state. The closure properties for activities ensure that no combinations of events and system state can be omitted from the definition of a process. The closure property for trigger expressions ensures that the result of an execution can also not be omitted. The result is that the specification is complete. It is important to note that it is very complex to define the ordering of activities and activity groups. We plan to address this problem in future through graphical user interface.

We have carried out an extensive case study based on an e-procurement application. This work shows that our model enables programmers to write the individual services participating in a distributed application in such a way that they deliver consistent outcomes despite concurrency, exceptions and failures. We have also tested our model by developing operational systems based on .NET technology. The insight gains from the prototype implementation leads us to further work on isolation, the classification of states and description of external interface (choreography).

References

1. Business Process Execution Language for Web Services (BPEL4WS), Version 1.0. <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>
2. Web Service Choreography Interface (WSCI) 1.0 Specification. <http://wwws.sun.com/software/xml/developers/wsci/>
3. G. Alonso, A. El Abbadi, M. Kamath, R. Gunthor, D. Agrawal, and C. Mohan. Failure handling in large scale workflow management systems. 1994, Technical Report IBM RJ9913.
4. G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Gunthor, and C. Mohan. Advanced transaction models in workflow contexts. IEEE Conference on Data Engineering, pp 574-581, 1996.
5. F. Casati, S. Ceri, S. Parboschi and G. Pozzi. Specification and Implementation of Exceptions in Workflow Management Systems. ACM Transactions on Database Systems, vol. 24, no. 3, pp. 405-451, 1999.
6. S. Ceri, P. Grefen, G. Sanchez. WIDE – a distributed architecture for workflow management. Proc RIDE'97, pp 76-81, 1997.
7. U. Dayal, M. Hsu, and R. Ladin. Organizing Long-Running Activities with Triggers and Transactions. Proc ACM International Conference on Management of Data (SIGMOD),, pp. 204-214, 1990.
8. J. Eder and W. Liebhart. The Workflow Activity Model WAMO. Proceedings of the 3rd International Conference on Cooperative Information Systems, Vienna, Austria, 1995.
9. H. Garcia-Molina and Salem, K., "Sagas," ACM International Conference on Management of Data (SIGMOD),, pp. 249-259, 1987.
10. D. Georgakopoulos, M. F. Hornick, and A. P. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure," Distributed and Parallel Databases, vol. 3, no. 2, pp. 119–153, 1995.
11. A. Geppert, D. Tombros, and K. Dittrich. Defining the Semantics of Reactive Components in Event-Driven Workflow Execution with Event Histories. Information Systems 23(3/4):235-252, 1998.
12. J.B. Goodenough. Exception Handling Issues and a Proposed Notation. Communications of the ACM, 18(12):683-696, 1975.
13. J. Gray and A. Reuter. Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publishers, 1993.
14. P. Greenfield, A. Fekete, J. Jang, and D. Kuo. Compensation is Not Enough. In 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC'03), September 2003, pp. 232-239, Brisbane, Australia
15. C. Hagen and G. Alonso. Exception Handling in Workflow Management Systems. IEEE Transactions on Software Engineering, 26(10):943-958, October 2000.
16. C. Hagen and G. Alonso. Flexible Exception Handling in the OPERA Process Support System. Proc IEEE International Conference on Distributed Computing Systems, pp 526-533, 1998.
17. C. Hagen and G. Alonso. Beyond the Black Box: Event-Based Inter-process Communication in Process Support Systems. Proc IEEE International Conference on Distributed Computing Systems, pp 450-457, 1999.
18. G. Knolmayer, R. Endl, and M. Pfahrer. Modeling Processes and Workflows by Business Rules. In Business Process Management, ed W. van der Aalst, LNCS 1806, pp 16-29, 2000.

19. D. Kuo, A. Fekete, P. Greenfield, and S. Nepal. Consistency for Service-Oriented Systems, Technical Report 05/017, CSIRO ICT Centre, Australia, 2005.
20. A. Koschel, and R. Kramer: Applying Configurable Event-triggered Services in Heterogeneous, Distributed Information Systems. Engineering Dederated Information Systems, Proceedings of the 2nd Workshop EFIS'99, May 5-7, 1999, Kühlungsborn, Germany,pp. 147-157
21. Z. Luo, A. Sheth, K. Kochut, and J. Miller. Exception Handling in Workflow Systems. Applied Intelligence 13(2):125-147, September 2000.
22. H. Mourani and P. Antunus. Exception Handling Through a Workflow. CoopIS 2004, pp. 37-54.
23. R. Muller, U. Greiner, and E. Rahm. AGENTWORK: a workflow system supporting rule-based workflow adaptation. Data and Knowledge Engineering, 51(2):223-256, November 2004.
24. S. Nepal, A. Fekete, P. Greenfield, J. Jang, D. Kuo and T. Shi. A Service-oriented Workflow Language for Robust Interacting Applications. CSIRO ICT Centre Technical Report 05/091, 2005.
25. S. D. Urban, S. Kambhampati, S. W. Dietrich, Y. Jin, A. Sundermier, An Event Processing System for Rule-Based Component Integration, Proceedings of the International Conference on Enterprise Information Systems, Portugal, April, 2004. pp.312-319.
26. D. Tombros and A. Geppert. Building Extensible Workflow Systems using an Event-Based Infrastructure. In Proc CAiSE'00, pp 325-339
27. H. Wächter and A. Reuter. The ConTract Model. In: Database Transaction Models for Advanced Applications, ed. A. K. Elmagarmid. Morgan Kaufmann, 1992.
28. J. Widom and S. Ceri, Active Database Systems: Trigger and Rules For Advances Database Processing, Morgan Kaufmann, 1995.

Balancing Flexibility and Security in Adaptive Process Management Systems

Barbara Weber¹, Manfred Reichert², Werner Wild³, and Stefanie Rinderle⁴

¹ Quality Engineering Research Group, University of Innsbruck, Austria
`Barbara.Weber@uibk.ac.at`

² Information Systems Group, University of Twente, The Netherlands
`m.u.reichert@cs.utwente.nl`

³ Evolution Consulting, Innsbruck, Austria
`werner.wild@evolution.at`

⁴ Dept. Databases and Information Systems, University of Ulm, Germany
`rinderle@informatik.uni-ulm.de`

Abstract. Process-aware information systems (PAIS) must provide sufficient flexibility to their users to support a broad spectrum of application scenarios. As a response to this need adaptive process management systems (PMS) have emerged, supporting both ad-hoc deviations from the predefined process schema and the quick adaptation of the PAIS to business process changes. This newly gained runtime flexibility, however, imposes challenging security issues as the PMS becomes more vulnerable to misuse. Process changes must be restricted to authorized users, but without nullifying the advantages of a flexible system by handling authorizations in a too rigid way. This paper discusses requirements relevant in this context and proposes a comprehensive access control (AC) model with special focus on adaptive PMS. On the one hand, our approach allows the compact definition of user dependent access rights restricting process changes to authorized users only. On the other hand, the definition of process type dependent access rights is supported to only allow for those change commands which are applicable within a particular process context. Respective AC mechanisms will be key ingredients in future adaptive PMS.

1 Introduction

In order to support a broad spectrum of applications, process-aware information systems (PAIS) must provide sufficient flexibility at run-time [1,2]. First, PAIS should be quickly adaptable to changes of the real-world processes (e.g., due to business reengineering efforts) [3,4,5]. Second, during the execution of individual process instances users must be able to flexibly deviate from the pre-modeled process schema (e.g., by adding or skipping tasks). Such ad-hoc deviations may become necessary to deal with exceptional situations [1,6].

In response to these needs adaptive process management systems (PMS) have emerged during recent years. Examples include ADEPT [1], CBRFlow [7], METEOR [8] and WASA2 [9]. All these PMS aim at the flexible support of changes

at the process type and/or the process instance level. This newly gained flexibility, however, imposes challenging security issues as the PMS becomes more vulnerable to misuse. For example, the uncontrolled addition of long-running activities to an ongoing process instance may delay the execution of the whole process. Appropriate access control (AC) mechanisms are thus even more important for adaptive than for traditional PMS [10,11,12,13,14,15,16] to avoid such misuse scenarios. Process changes must be restricted to authorized users only, but without nullifying the advantages of a flexible system by handling authorizations in a too rigid way.

Although there are several approaches for AC in traditional process management systems (PMS), the special requirements of adaptive PMS have not been addressed in a sufficient way so far. Existing approaches either assume that process schemes are modeled only once and then remain unchanged (i.e., covering access rights for the execution of tasks only) (e.g., [13,10]) or they only support the definition of change rights at a very coarse-granular level [17]. This restricted view, however, is not applicable to adaptive PMS, which require adequate access rights for the different kinds of changes. In particular, access rights must be simple to define, easy to maintain and fast to check.

In addition, it must be possible to enforce the execution of particular activities (e.g., due to legal requirements) and to ensure that only activities which are applicable in a specific context can be inserted into a process instance. For a drug procurement process in a hospital, for instance, the insertion of a patient treatment step makes no sense and should thus not be allowed.

Defining process changes requires user experience and is error prone if not supported by suitable tools. Therefore, adaptive PMS should assist the user while performing changes by displaying only those change commands which are applicable in the current context and for which the user holds the necessary access rights.

In the past we developed detailed concepts for the dynamic modification of process instances, for the evolution of process types and for the memorization and the reuse of process instance changes. This work has been done in the ADEPT and CBRFlow projects [1,7,18,19,20], security issues have not been considered in detail so far. In this paper we introduce an advanced AC model which covers the particular requirements of adaptive PMS. The paper is organized as follows: Section 2 covers adaptive PMS and their characteristics. Section 3 describes the requirements for an AC model, Section 4 extends the core RBAC model to meet the specific requirements for adaptive PMS. Section 5 adds process type dependent access rights to the extended RBAC model. Section 6 describes practical issues and Section 7 discusses related work. The paper concludes with a summary and an outlook in Section 8.

2 Adaptive PMS and Their Characteristics

This section describes background information on adaptive PMS as needed for the further understanding of this paper.

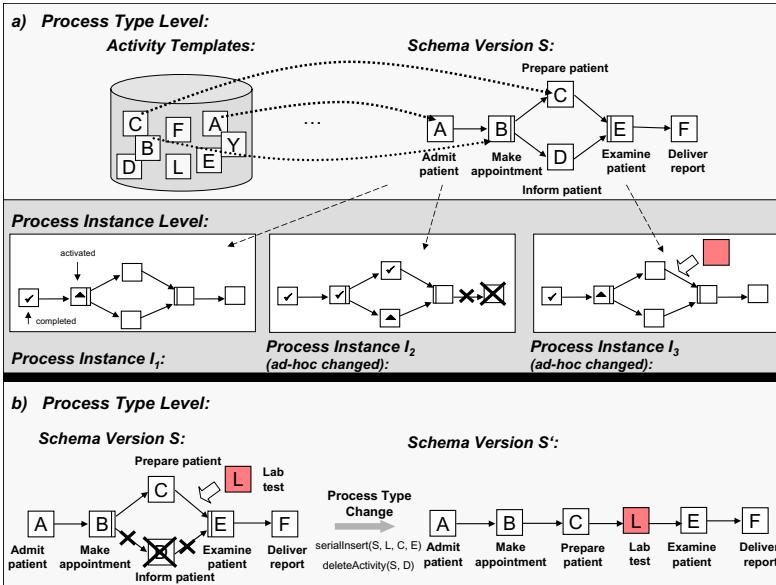


Fig. 1. Different Levels of Process Change (Clinical Example)

2.1 Basic Concepts

In a PMS, for each supported business process (e.g., booking a business trip or handling a medical order) a *process type* T has to be defined. For a particular type one or more *process schemes* (process templates) may exist reflecting different schema versions of T . In Fig. 1 (b), for example, S and S' correspond to different schema versions of the same type. A process schema itself is represented by a directed graph, which consists of a set of *activities* a_1, \dots, a_n and the *control edges* connecting them. Process schema version S from Fig. 1 (b) consists of six activities; Activity *Admit patient* is followed by activity *Make appointment* in the flow of control, whereas *Prepare Patient* and *Inform Patient* can be processed in parallel. Each activity is based on a predefined activity template, which is maintained and stored in a repository. In general, a particular activity template can be reused within different process schemes.

Based on a schema S new *process instances* I_1, \dots, I_m can be created and executed at runtime. In Fig. 1 (a), for process instance I_1 activity *Admit patient* has already been completed whereas activity *Make appointment* is currently activated (i.e., it is offered to users in their worklists).

2.2 Process Change

Adaptive PMS are characterized by their ability to correctly and efficiently handle process changes. In general, changes are triggered and performed at two levels – the process type and the process instance level.

Changes to a process type T may become necessary to cover the evolution of real-world business processes [3,4,9]. Process engineers can accomplish a type change by applying a set of change commands to the current schema version S of type T . This results in the creation of a new schema version S' of the same type (cf. Fig. 1 a). Execution of future process instances is then based on S' . For long-running processes it might be necessary to migrate already running process instances to the new schema version S' [18].

By contrast, ad-hoc changes of individual process instances are usually performed by process participants (i.e., end users). Ad-hoc changes are necessary to react to exceptions or unanticipated situations [1,7,8]. The effects of such instance-specific changes are kept local, i.e., they do not affect other process instances of the same type. In Fig. 1 (a) instance I_2 has been individually modified by dynamically deleting activity `Deliver report`. Thus the execution schema of I_2 deviates from the original process schema S of this instance.

In order to facilitate exception handling, adaptive PMS should allow for the memorization and the reuse of ad-hoc deviations. For this, our approach applies case-based reasoning techniques [19,21]. More precisely, changes of individual process instances can be performed either by explicitly defining the change from scratch or by reusing information about previous changes (which were successfully applied to other process instances in similar problem situation before).

In our approach both process type and process instance changes are based on a complete set of change commands with well-defined semantics [1,22]. Table 1 presents selected *high-level change commands* provided in this context.

Table 1. A Selection of ADEPT Change Commands*

Change Command applied to Schema S	Effects on Schema S
Additive Change Commands	
serialInsert(S, X, A, B)	insert activity X into schema S between the two directly connected activities A and B
parallelInsert(S, X, A)	insert activity X into schema S parallel to activity A
Subtractive Change Commands	
deleteActivity(S, X)	delete activity X from schema S
Order-Changing Commands	
serialMove(S, X, A, B)	move activity X from its current position in schema S to the position between two directly connected activities A and B

*A detailed description of all change commands supported by ADEPT can be found in [18,22].

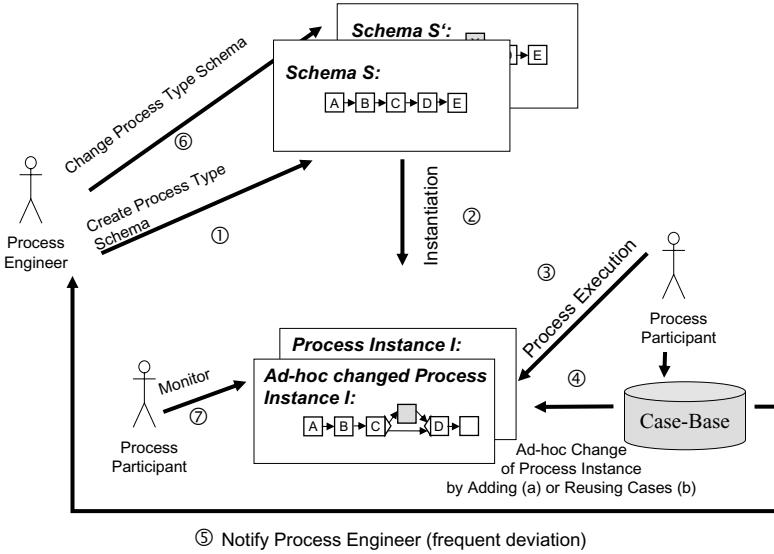


Fig. 2. Major Use Cases for an Adaptive PMS

2.3 Major Use Cases for an Adaptive PMS

In order to construct a comprehensive AC model for adaptive PMS we must focus on the major use cases of such a system in detail [20]. An overview is given in Fig. 2. At buildtime an initial computerized representation of a company's business processes is created either by business process analysis or by applying process mining techniques (i.e., by observing process and task executions) (1). At run-time new process instances are created from these predefined process schemes (2). In general, process instances are then executed according to the process type schema they were derived from and activities are allocated to authorized process participants to perform the respective tasks (3). However, when deviations from the predefined schema become necessary at the process instance level (e.g., due to exceptions), process participants must be able to deviate from it. They can either specify a new ad-hoc deviation and document the reasons for the changes in a case-base (4 a), or they can reuse a previously specified ad-hoc modification from the case-base (4 b). The PMS monitors how often a particular schema is instantiated and how frequently deviations occur. When a particular ad-hoc modification is frequently reused, the process engineer is notified that a process type change should be performed (5). The process engineer can then evolve the process type schema, and, as far as possible, migrate running instances to the new schema version (6). During run-time process instances can be monitored by process participants (7). Finally, in addition to these use cases, all PMS must support granting access rights.

3 Requirements for an AC Model for Adaptive PMS

To our best knowledge most existing AC models for PMS [10,11,12,13,14,15,16] ignore the problem of process change and therefore do not meet the specific requirements posed by adaptive PMS (cf. Section 7). In this section specific requirements for AC models in adaptive PMS are elaborated. All requirements stem from real world case studies in the medical domain [23].

Requirement 1 (Support of user dependent and process type dependent access rights). An AC model for adaptive PMS should support the definition of both *user dependent* and *process type dependent* access rights in an integrated way. While the former restrict access to authorized users in order to avoid misuse (e.g., only users with role *physician* are authorized to insert the *X-ray* activity), the latter are applied to only allow for change commands that are useful within a particular context (e.g., activity *vacation request* must not be inserted in medical treatment processes).

Requirement 2 (Completeness of the AC model). In order to adequately support adaptive processes it is not sufficient to only provide access rights for executing activities. In addition, all presented use cases (cf. Fig. 2) must be covered. Furthermore, the rights to change process types and process instances must be granted separately, as, for example, a user who is allowed to change a specific process instance usually is not authorized to change the process type schema as well. As another example consider the introduction of a process instance change by reusing information about a previously defined ad-hoc modification: authorization for this use case does not necessarily imply that respective users are also authorized to define a new ad-hoc change from scratch.

Requirement 3 (Fine-grained definition of access rights). In general, it must be possible to specify access rights for individual change commands or groups of change commands (e.g., a particular role is only allowed to insert additional process activities, but not to delete existing ones). In any case, an AC model for adaptive PMS must allow the definition of access rights for all change commands and their parameterizations. For example, a *physician* is authorized to insert additional activities. However, this authorization may be restricted to medical treatment activities (e.g., X-ray, Computer Tomography) and selected processes (e.g., patient examination).

Requirement 4 (Usability and maintainability of access rights). A significant challenge is to balance flexibility and security in such a way that the advantages provided by adaptive PMS are not nullified by a too rigid AC model. Thus, access rights themselves should be simple to define and easy to maintain. In order to support the easy and compact definition of access rights, objects should be hierarchically composed and allow for the definition of access rights at different levels of granularity. For instance, it might be reasonable to authorize a particular role to perform process type changes for all process type schemes supported by the PMS. However, in corporations with a large number of process type schemes different users might be responsible for individual process type schemes or for groups of process type schemes.

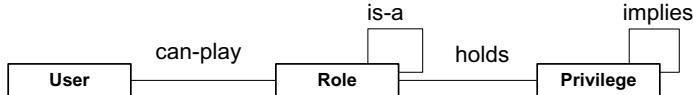


Fig. 3. Core Access Control Model

4 User Dependent Access Rights

An AC model for (adaptive) PMS must allow the system administrator to restrict access to authorized users to avoid misuse. In Section 4.1 we first review basic properties of the core RBAC model, in Section 4.2 and Section 4.3 we then derive an extended role-based AC model to meet the specific requirements of adaptive PMS.

4.1 Core AC Model

The RBAC model is frequently used to specify access rights in PMS [24,25,26]. Access rights are not directly linked to concrete users, but to the more abstract concept of a role. Such roles group privileges (i.e., classes of access rights) and are assigned to users based on their capabilities and competences. *Physician*, *nurse*, and *technician* are examples for roles in a hospital. The role *physician* may include the privileges to order a lab test or to perform a medical examination. Users possessing the same role are considered as being interchangeable, i.e., all users holding a particular role qualify for the privileges associated with that role. A user can hold several roles. In addition, roles can be hierarchically organized, which allows to (transitively) propagate the privileges associated with a particular role to the more specific roles. A head nurse, for instance, is a nurse and therefore inherits all privileges of role *nurse*. Finally, privileges themselves are organized hierarchically to foster inheritance of access rights. The privilege to order restricted drugs (e.g., morphine) up to quantity 1000 implies the more restricting privilege to order such drugs up to quantity 50. Formally, a RBAC model [10] consists of a set of users U , a set of roles R , and a set of privileges P as well as the relationships between elements of these sets (cf. Fig. 3).

In the following the entities and relationships from Fig. 3 are described in more detail.

- A user $u \in U$ represents an individual actor.
- A role $r \in R$ denotes a grouping of privileges which can be assigned to one or more users.
- A privilege $p \in P$ represents a class of rights, e.g., to perform certain operations or to access certain data.
- $can-play(u, r)$ states that user u holds role r .
- $is-a(r_1, r_2), r_1, r_2 \in R$ states that role r_1 specializes role r_2 and thus inherits all privileges from r_2 .

- $\text{holds}(r, p), r \in R, p \in P$ states that role r holds privilege p .
- $\text{implies}(p_1, p_2), p_1, p_2 \in P$ states that privilege p_1 includes privilege p_2 .

4.2 Extended Access Control Model

In this section we extend the core RBAC model by adding additional entities and relationships to construct an adequate AC model for adaptive PMS, which meets Requirements 1-4 of Section 3.

Operation. AC models for adaptive PMS must cover all use cases from Fig. 2 (e.g., changing processes, executing process activities, monitoring process instances, etc.) and be able to grant access rights to each of them separately. We therefore add the entity *operation* to the core RBAC model introduced above.

As illustrated in Fig. 4 (a) we distinguish between seven major operations: *ChangeProcess*, *CreateSchema*, *ExecuteActivity*, *GrantPrivilege*, *InstantiateSchema*, *MonitorProcessInstance* and *NotifyUser*. The abstract operation *ChangeProcess* is further divided into process type and process instance changes. Process instance changes can further be split into two groups depending on whether information about a previously defined ad-hoc modification is reused or a new change has to be defined from scratch.

Operations are hierarchically organized, so privileges for a particular operation are automatically extended to their descendants as well (*include* relationship in Fig. 7). As illustrated in Fig. 4 (a), a user holding the privilege for abstract operation *ChangeProcess* is authorized to perform changes at both the process type and the process instance level. The latter can be handled either by adding new or by reusing existing ad-hoc change cases. In contrast, the operation *ReuseExistingProcessInstanceChange* only authorizes for process instance changes based on the reuse of previously defined ad-hoc modifications, but not to define new ad-hoc changes.

Change Command. Although the concept *operation* allows for separate access rights for process type and process instance changes, it does not differentiate between change commands. However, this is indispensable for AC in adaptive PMS as a user might be authorized to skip an activity, but not to perform structural changes by inserting an additional activity. Therefore, we further extend our model with the entity *change command*.

In adaptive PMS both process type and process instance changes can be accomplished by applying a set of well-defined change commands (cf. Fig. 4 b)¹. When applying the command *serialInsert* an additional activity is inserted between two succeeding process activities. For skipping a particular activity, for example, the command *deleteActivity* can be used. Using different kinds of change commands requires different levels of user experience, which should be taken into account when defining privileges for process change. For example, the deletion of a particular process activity during runtime is always limited in scope and

¹ For illustration we use the ADEPT change commands; however, the model is applicable to other command sets as well.

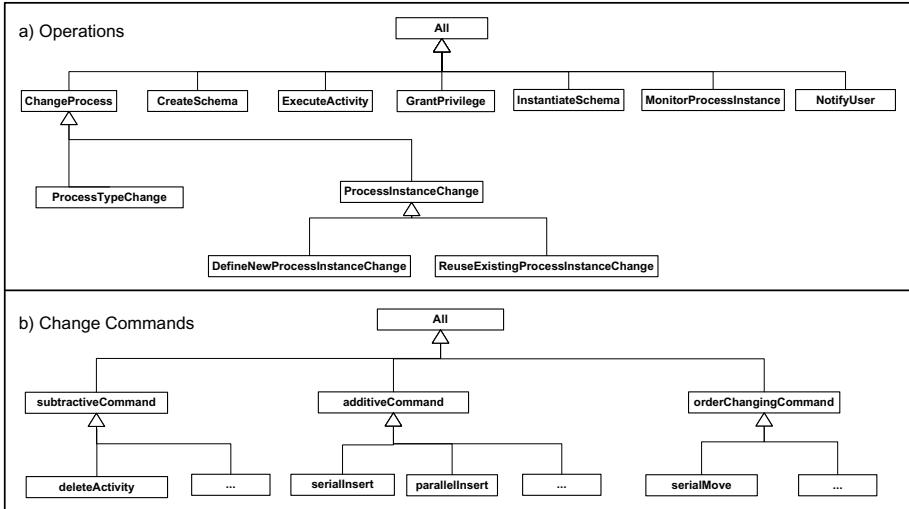


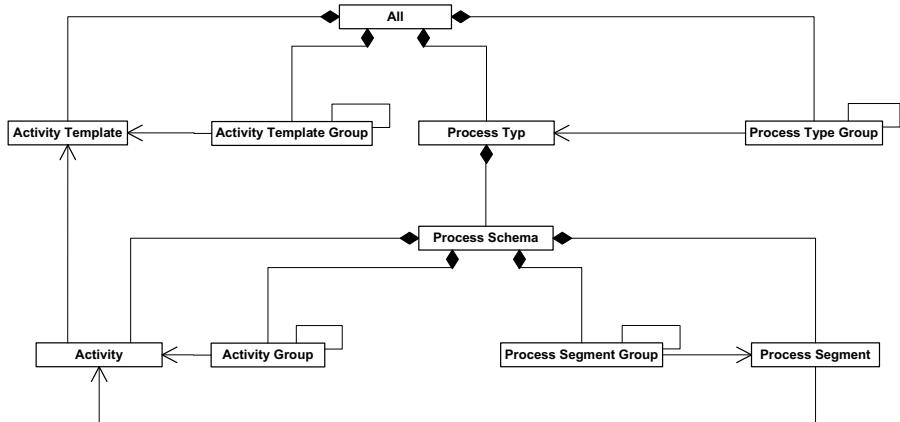
Fig. 4. Operations and Change Commands

can therefore easily be accomplished by end users (i.e., only a few parameters have to be specified when defining the change). In contrast, the insertion of an activity usually requires more comprehensive parameter specifications (e.g., parameters specifying the position of the newly inserted activity) and is therefore more complex to handle.

In order to be able to define access rights at different levels of granularity we allow for the hierarchical organization of change commands (*specializes* relationship in Fig. 7). As illustrated in Fig. 4 (b), the abstract change command *All* includes all kinds of subtractive, additive and order changing commands. If a user holds the right to perform this abstract change command he automatically inherits the right to perform all change commands lower in the hierarchy as well (e.g., *serialInsert*, *deleteActivity*). On the one hand this approach allows us to define privileges in a very compact way. On the other hand we are able to provide fine-grained specifications as well.

Object. So far we have only considered operations and different change commands. However, we not only must be able to express that a certain role is allowed to delete or add process activities, but also to state which object a particular operation or change command uses (e.g., the process instance that may be monitored or the activity that may be added by an insert command), i.e., the parameterization of operations and change commands has to be considered as well. Therefore we introduce the entity *object* as another dimension in our AC model (cf. Fig 5). For instance, while examining a patient a physician can insert an additional activity *X-ray* (*object*).

Objects are hierarchically organized to achieve maintainable models (*containsObject* relationship in Fig. 7). As illustrated in Fig. 5 access rights can be

**Fig. 5.** Object Hierarchy

defined, for instance, for the whole PMS (least detailed level) down to the granularity of single process types or activities (most detailed level). As illustrated in Fig. 6 a particular role holds the privilege to perform the *deleteActivity* command for process type T_1 (=object). Thus this role is authorized to delete all activities which are part of process schemes related to T_1 , i.e., activities a_{11} , a_{12} , a_{21} and a_{311} .

Note that a particular operation may not be applicable to all kinds of objects. Table 2 summarizes which combinations of operations and objects are applicable. For example, the *ChangeProcess* operation in combination with an order-changing change command can be used with the following objects: process

Table 2. Granularity of Objects Depending on the Operation*

Operation	All	T	PTG	S	PS	PSG	A	AG	AT	ATG
Change Process										
- additive change								X	X	
- subtractive change	X	X	X	X	X	X	X	X	X	X
- order-changing change	X	X	X	X	X	X	X	X	X	X
Create Schema	X		X							
Execute Activity							X	X		
Grant Privilege	X	X	X	X	X	X	X	X	X	X
Instantiate Schema	X	X	X	X						
Monitor Process Instance	X	X	X	X	X	X				
Notify User	X	X	X	X	X	X				

*All = Process Management System, T = Process Type Schema, PTG = Process Type Group,

S = Process Schema Version, PS = Process Segment, PSG = Process Segment Group,

A = Activity, AG = Activity Group, AT = Activity Template, ATG = Activity Template Group.

management system (*All*), process type, process type group, process schema version, process segment, process segment group, activity and activity group. In contrast, for the *ChangeProcess* operation in combination with additive change commands only activity templates or groups of activity templates can be used as the object.

Subject. Finally, we introduce the entity *subject* to specify what is subject to change. For example, when an additional lab test activity (*object*) is inserted (*change command*), we must know the process schema version or the process segment it will be added to, i.e., the subject for this insertion command must be specified.

Like objects, subjects are organized in a hierarchy too (*containsSubject* relationship in Fig. 7). When specifying subjects only a sub-set of the elements from Fig. 5 is required. Candidates for subjects are the whole process management system, process types, groups of process types, process schema versions and process segments; however, activities, groups of activities, activity templates and groups of activity templates are not applicable.

As illustrated in Fig. 6, if a particular role has the privilege to perform process instance changes (=operation) using the *serialInsert* command for activity template at_1 (=object) and subject T_1 , then this role is authorized to insert activity template at_1 into process instances based on process type schema T_1 .

Specifying a subject is only needed for *additive* change commands. For all other operations no subject must be specified as the subject is automatically known by the system, due to the containment relationships (cf. Fig. 5). When, as illustrated in Fig. 6, the object for operation *ExecuteActivity* is activity a_{11} , then the subject is implicitly known to be S_1 , the process schema version activity a_{11} is part of.

Privilege. As illustrated in Fig. 6 a privilege is defined by specifying an operation, an object, a change command, and a subject. For example, privilege (*ProcessInstanceChange*, at_1 , *serialInsert*, T_1) states that activities derived from activity template at_1 can be added to any process instance created from process type T_1 . Fig. 6 shows the field values of the selected privilege in dark grey. The selected privilege automatically extends to the light grey boxes as well; for example, in Fig. 6 the subject is T_1 , this extends the privilege to process schema versions S_1 and S_2 too.

As indicated in Fig. 6 not all entities are mandatory in all situations. The entities *operation* and *object* are always mandatory, while the entity *change command* is only mandatory when the *ChangeProcess* operation is selected. The entity *subject* is only mandatory for *additive* change commands.

4.3 Extended Model - Overview

We added the additional entities *operation OP*, *object O*, *change command C* and *subject SUB* as well as the relationships *includes*, *containsObject*, *specializes* and *containsSubject* to the core RBAC model. In addition the entity *privilege*

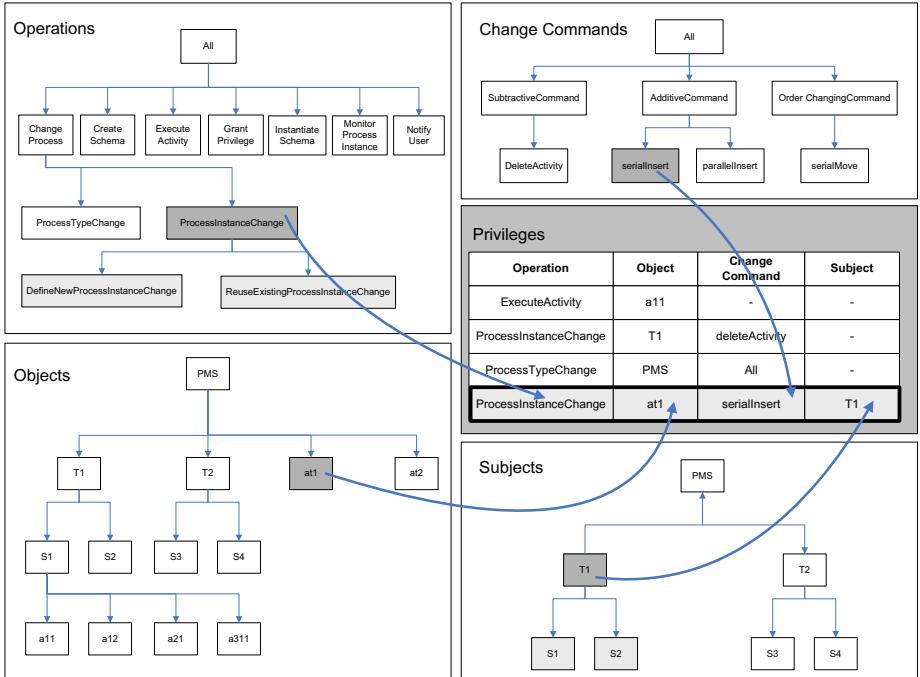


Fig. 6. Extended Access Control Model - Example

P has been defined in a more detailed way. The meta-model of the extended AC model is illustrated in Fig. 7.

- Object $o \in O$ represents one of the following entities: the entire process management system, a process type, groups of process types, a process schema, a process segment, groups of process segments, an activity, a group of activities, an activity template or a group of activity templates.
- Operation $op \in OP$ represents a use case supported by the adaptive PMS (e.g., *ExecuteActivity*, *GrantPrivilege* or *ChangeProcess*).
- Change Command $c \in C$ represents a change command (e.g., *deleteActivity* or *serialInsert*).
- Subject $sub \in SUB$ represents what is subject to change, i.e., the entire process management system, a process type, a group of process types, a process schema, a process segment and a group of process segments.
- Privilege $p \in P$ is a tuple (op, o, c, sub) representing the right to perform a particular operation op with an object o using change command c on a subject sub .
- $\text{includes}(op_1, op_2), op_1, op_2 \in OP$ states that operation op_1 includes op_2 . Having a privilege for op_1 thus includes the privileges for op_2 .
- $\text{containsObject}(o_1, o_2), o_1, o_2 \in O$ states that object o_1 includes object o_2 .

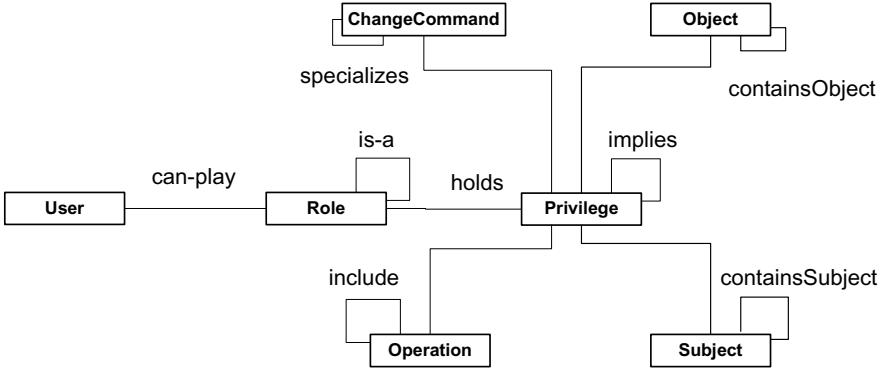


Fig. 7. Extended Access Control Model

- $\text{specializes}(c_1, c_2), c_1, c_2 \in C$ states that change command c_1 includes change command c_2 .
- $\text{containsSubject}(sub_1, sub_2), sub_1, sub_2 \in Sub$ states that subject sub_1 includes subject sub_2 .

5 Process Type Dependent Constraints

An AC model for adaptive PMS must not only allow to restrict access to authorized users. It must also enforce the execution of particular activities (e.g., a particular activity must not be deleted due to legal requirements) and ensure that only semantically correct activities can be inserted in the given context (e.g., no patient related activities must be inserted into a drug procurement process).

Process type dependent access rights allow to specify which activities, activity templates, which groups of activities and which groups of activity templates can be inserted into, deleted from or moved within process instances based on a particular process type.

Process type dependent access rights are defined independently of the individual users and roles performing the process instance change, thus only a sub-set of the extended AC model in Fig. 7 is needed; the entities *role* and *user* can be omitted. Like user dependent access rights, a privilege for process type dependent access rights consists of the entities *operation*, *object*, *change command* and *subject*.

Example 1. Privilege (*ProcessInstanceChange*, *MedicalTreatmentStep*, *additive-Command*, *GroupMedicalTreatmentProcess*) says that any activity template of group *MedicalTreatmentStep* (i.e., X-ray, Lab Test, Computer Tomography) can be inserted into any medical treatment process by using an additive change command.

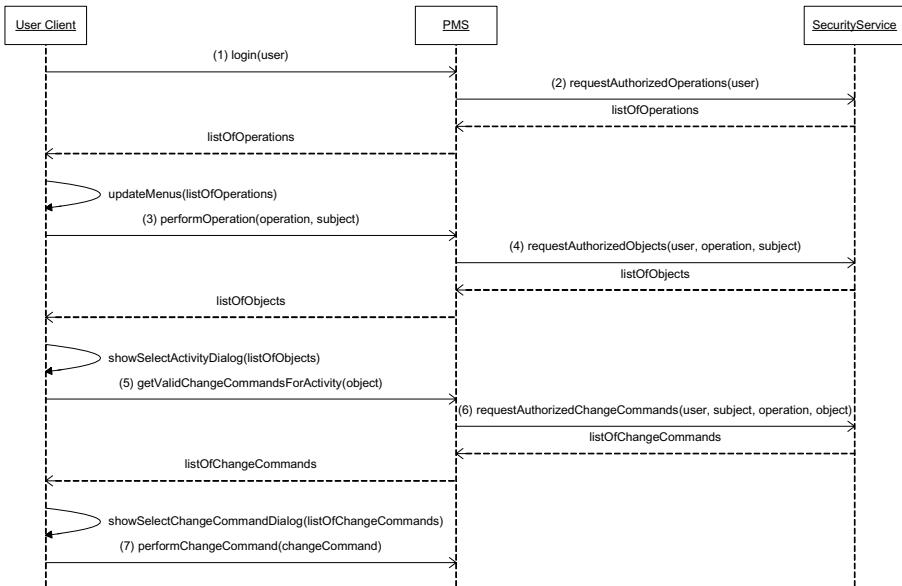


Fig. 8. PMS and Security Service Interactions (Process Instance Change)

6 Practical Issues

Our extended AC model is currently implemented as a separate Security Service (SECS) which can be used independently of a specific PMS. To demonstrate the interactions between a PMS and the SECS this section walks the reader through a process instance change (cf. Fig. 8).

Dynamic Menu Configuration. When the user logs in to the PMS via his user client (1) the PMS interacts with the SECS to request the list of operations the user is authorized for (2) (Query: *requestAuthorizedOperations(User)*). Based on the results of this query the menu of the user client is dynamically configured, i.e., only those operations are displayed in the user's menu for which he is authorized. Thus, the user is never presented a menu item he is not authorized for, which prevents annoying "not authorized" warnings.

Example 2. The physician *John* logs in to the PMS via his user client. The PMS then dynamically configures the menu items based on the privileges which have been assigned to the role *physician* and which comply with the process type dependent access rights (cf. Fig. 9 a). John, for instance, is authorized to insert activity templates of group *MedicalTreatmentSteps* (i.e., *X-ray*, *Lab Test*, *Computer Tomography*) into process instances of *medical treatment processes*.

Perform Ad-hoc Changes. When an authorized user wants to deviate from the predefined process schema at runtime he can select the respective menu item to perform a process instance change in his user client (3).

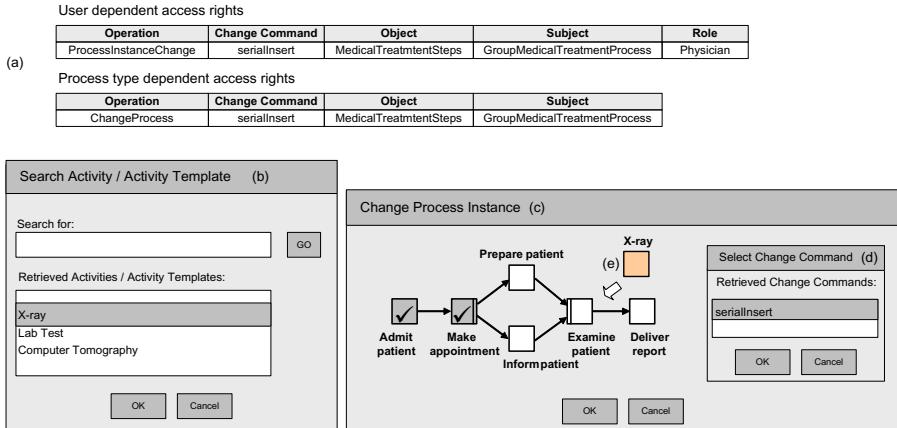


Fig. 9. Example Privileges (a) and User Interactions (b-e)

He then has to select the object for the change, e.g., the activity template to be inserted into the process instance schema or the concrete activity to be deleted or moved (cf. Fig. 9 b).

For insert operations the PMS requests the set of activity templates the user is authorized to insert into process instances based on process schema version S (4). It further requests the set of activities in process schema S the user is authorized to delete or move and which comply with the process type dependent access rights (Query: *requestAuthorizedObjects(User, Subject, Operation)*). Completed activities are not listed, as already passed process graph regions can no longer be modified. The user can then select one of the displayed activities or activity templates (5).

Example 3. Assume that John decides to perform a new ad-hoc modification (e.g., to insert a new activity based on template X-ray) for a process instance created from process schema version S1 (cf. Fig. 9 b). He selects the respective operation in the menu bar of his user client, and the system then requests the list of activity templates (*requestAuthorizedObjects(John, S1, ProcessInstanceChange)*) he is authorized to add to this schema. In our example, the SECS returns activity templates X-ray, Lab Test, and Computer Tomography.

The system then shows a graphical representation of process schema S (cf. Fig. 9 c) and suggests those change commands to the user which comply with both his authorizations and the process type dependent access rights (6) (Query: *requestChangeCommands(User, Operation, Object, Subject)*). The user can then choose one of the displayed change commands for execution (7). When necessary, the system requests any required parameters from the user (cf. Fig. 9 d).

Example 4. John selects activity X-ray. The PMS then displays S1, the process schema on which the process instance to be modified was created from.

After this, the PMS requests a list of change commands which are presented to John ($\text{requestChangeCommands}(\text{John}, \text{ProcessInstanceChange}, \text{X-ray}, S1)$). John then selects the *serialInsert* change command to insert the additional X-ray activity. Furthermore, the system asks him where the X-ray activity should be inserted and ensures that he does not insert it into already passed process graph regions. John replies that the X-ray activity is to be performed after activity **Examine Patient** and before activity **Deliver Report** (cf. Fig. 9 e).

7 Related Work

There are several AC models for PMS discussed in literature [10,11,12,13,14,15, 16,17]. Most of them use RBAC models and support static as well as dynamic constraints (e.g., [10,11,12,13,14]). However, they only provide limited support for adaptive processes: either they only cover privileges for the execution of tasks, but do not deal with privileges for process changes at all (e.g., [10,13]), or specify change rights at a very coarse-grained level (e.g., not distinguishing between change commands) [17].

W-RBAC [10] provides a specific AC model for workflow systems which focuses on the definition of task execution rights and allows for the definition of static and dynamic constraints (e.g., separation of duties). Dynamic constraints have not been the focus of this paper, but will be considered in our overall approach as well, we plan to apply concepts provided by existing approaches [10,11,12,13,14]. In order to deal with exceptional situations W-RBAC allows authorized users to override constraints. However, as the definition of change rights is not supported, W-RBAC is not suited for adaptive PMS.

Case-handling systems provide an alternative to adaptive PMS. FLOWER [15], for example, allows defining change rights to some degree. For each process and for each activity an *execution role* (for carrying out the activity or to start the process), a *redo role* (to undo activities) and a *skip role* (to omit an activity) can be defined. The handling of more complex change commands (e.g., the insertion of new activities) is not addressed.

Domingos et al. [17] propose an AC model for adaptive workflows, which also considers the evolution of access rights. Though their approach differentiates between process type and process instance change, it does not allow for fine-grained definition of privileges at the level of individual change commands. Their focus is on user dependent access rights, process dependent rights are not considered. As no abstraction mechanisms (e.g., hierarchies) are used, the compact definition of access rights is not possible.

An approach to control flexibility other than by AC is the Pockets of Flexibility model [27]. Flexibility is limited to certain predefined points in the workflow, where the workflow can be dynamically extended at run-time.

8 Summary and Outlook

In this paper we presented an AC model which allows for the compact definition of access rights as needed in adaptive PMS. We support both the definition of user dependent and process type dependent access rights (cf. Requirement 1). Our approach supports the use cases provided by an adaptive PMS (cf. Requirement 2), and allows the specification of access rights for individual change commands (cf. Requirement 3). If desired, access rights can be specified at an abstract (i.e., coarse-grained) level by using the hierarchical organization of our model. Fine-grained specification of access rights is supported as well, allowing context-based assistance of users when performing a change. However, the more detailed the respective specifications, the more costly their definition and maintenance becomes. Based on our experience with processes from several application domains, different granularities must be supported (cf. Requirement 4).

Currently we are working on the implementation of the Security Service and its integration into the adaptive PMS ADEPT [1] and CBRFlow [7]. We further plan a thorough evaluation of the AC model in real world settings, including its performance and scalability. Next, dynamic constraints will be elaborated in more detail and integrated into our AC model.

References

1. Reichert, M., Dadam, P.: ADEPT_{flex} - supporting dynamic changes of workflows without losing control. *JIIS* **10** (1998) 93–129
2. Jørgensen, H.D.: Interactive Process Models. PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway (2004)
3. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. *Data and Knowledge Engineering, Special Issue on Advances in Business Process Management* **50** (2004) 9–34
4. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. *Data and Knowledge Engineering* **24** (1998) 211–238
5. v.d. Aalst, W., Basten, T.: Inheritance of workflows: An approach to tackling problems related to change. *Theoret. Comp. Science* **270** (2002) 125–203
6. Strong, D., Miller, S.: Exceptions and exception handling in computerized information processes. *ACM–TOIS* **13** (1995) 206–233
7. Weber, B., Wild, W., Breu, R.: CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning. In: Proc. European Conf. on Case-based Reasoning (ECCBR'04), Madrid (2004) 434–448
8. Luo, Z., Sheth, A., Kochut, K., Miller, J.: Exception handling in workflow systems. *Applied Intelligence* **13** (2000) 125–147
9. Weske, M.: Workflow management systems: Formal foundation, conceptual design, implementation aspects. University of Münster, Germany (2000) Habil Thesis.
10. Wainer, J., Barthelmeß, P., Kumar, A.: W-RBAC - a workflow security model incorporating controlled overriding of constraints. *IJCIS* **12** (2003) 455–485
11. Bertino, E., Ferrari, E., Alturi, V.: The specification and enforcement of authorization constraints in wfms. *ACM Trans. on Inf. and Sys. Sec.* **2** (1999) 65–104
12. Botha, R., Eloff, J.: A framework for access control in workflow systems. *Information Management and Computer Security* **9** (2001) 126–133

13. Casati, F., Castano, S., Fugini, M.: Managing workflow authorization constraints through active database technology. *Inf. Sys. Frontiers.* **3** (2001) 319–338
14. Liu, D.R., Wu, M.Y., Lee, S.T.: Role-based authorization for workflow systems in support of task-based separation of duty. *The Journal of Systems and Software.* **73** (2004) 375–387
15. van der Aalst, W., Weske, M., Grünbauer, D.: Case handling: A new paradigm for business process support. *Data and Knowledge Engineering.* **53** (2005) 129–162
16. Atluri, V., Huang, W.K.: Enforcing mandatory and discretionary security in workflow management systems. *Journal of Computer Security.* **5** (1997) 303–339
17. Domingos, D., Rito-Silva, A., Veiga, P.: Authorization and access control in adaptive workflows. In: *ESORICS 2003.* (2003) 23–38
18. Rinderle, S.: Schema Evolution in Process Management Systems. PhD thesis, University of Ulm (2004)
19. Rinderle, S., Weber, B., Reichert, M., Wild, W.: Integrating process learning and process evolution - a semantics based approach. In: *BPM 2005.* (2005)
20. Weber, B., Reichert, M., Rinderle, S., Wild, W.: Towards a framework for the agile mining of business processes. In: *Proc. of Int'l BPI workshop.* (2005)
21. Weber, B., Rinderle, S., Wild, W., Reichert, M.: CCBR–driven business process evolution. In: *ICCBR'05, Chicago* (2005)
22. Reichert, M.: Dynamic Changes in Workflow-Management-Systems. PhD thesis, University of Ulm, Computer Science Faculty (2000) (in German).
23. Konyen, I.: Organizational structures and business processes in hospitals. Master's thesis, University of Ulm, Computer Science Faculty (1996) (in German).
24. Ferraiolo, D., Kuhn, D.: Role based access control. In: *15th National Computer Security Conference.* (1992)
25. Sandhu, R.S., Coyne, E., Feinstein, H., Youman, C.: Role-based access control models. *IEEE Computer* **29** (1996) 38–47
26. Ferraiolo, D.F., Chandramouli, R., Kuhn, D.R.: *Role-Based Access Control.* Artech House, Incorporated (2003)
27. Sadiq, S., Sadiq, W., Orlowska, M.: Pockets of flexibility in workflow specifications. In: *Proc. Int'l Entity–Relationship Conf. (ER'01), Yokohama* (2001) 513–526

Enabling Business Process Interoperability Using Contract Workflow Models

Jelena Zdravkovic^{1,2} and Vandana Kabilan²

¹ Department of Computer Science, University of Gävle
Kungsbacksvägen 47, 80 277 Gävle, Sweden

² Department of Computer and Systems Sciences, Stockholm University
and Royal Institute of Technology, Forum 100, SE-164 40 Kista, Sweden
{jzc, vandana}@dsv.su.se

Abstract. Business transactions are governed by legally established contracts. Contractual obligations are to be fulfilled by executing business processes of the involved parties. To enable this, contract terms and conditions need to be semantically mapped to process concepts and then analyzed for compliance with existing process models. To solve the problem, we propose a methodology that, using a layered contract ontology, deduces contract requirements into a high-level process description named Contract Workflow Model (CWM). By applying a set of transformation rules, the CWM is then compared for compliance with existing, executable process models. By the use of its concepts, the methodology enables comprehensive identification and evolution of requirements for interoperability of processes of the contracting parties.

1 Introduction

The purpose of establishing a business contract is to agree upon the rules and regulations governing interactions between the agreement partners. A contract testifies the legal binding nature of an agreement, as well as the remedial options and rights granted to the partners in case of any disagreement or dispute. Information contained in a contract needs to be assimilated as knowledge, which may be analyzed and re-used in the business process management.

The contract management domain has existed for some time as well as the business process management. However, most available solutions such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM) or other database applications for enterprise or contract management, have not managed to integrate the two domains seamlessly. A business contract is like a master plan for expected business behavior of the parties involved. Generally, it covers most contingencies and probable scenarios for planned execution of the commitments the parties make to each other. Thus, non-compliance to the contract terms could lead to legal, economic and business repercussions. This means the actual business processes of the parties must follow the rules established by a contract.

A successful contract is negotiated such that the terms are beneficial to all the parties involved. The contract stipulates both implied and explicit *obligations*, which are legal bindings, and need to be fulfilled as agreed. Therefore, a contract once negoti-

ated and signed cannot be forgotten. It needs to be realized by the business process. To ensure interoperability of the partners' business processes, it is necessary to have ability to examine the compliance between the process models and the respective contract requirements. Existing methodologies fail to bridge the chasm in between the two domains.

In this work, we first present a detailed conceptual model for modeling contract knowledge in a structured and layered form, named Multi Tier Contract Ontology. Secondly, we analyze different types of contract obligations and we identify the states through which each obligation passes through. Thirdly, based on the contract ontology and obligation state classification, we deduce a high-level process model named Contract Workflow Model (CWM). Lastly, by utilizing a structured process design framework, we set up a comprehensive set of rules for examining the compliance between CWM and existing executable business processes (or simply, *business processes*). The use of the rules enables continuous traceability of alignment between the contract obligations represented by a CWM, and a business process.

The paper is structured as follows. Section 2 gives an overview of related research. In Section 3 we first summarize our previous work on the Multi Tier Ontology; using a case study based on the INCOTERMS delivery patterns, we then introduce the methodology for deducing the Contract Workflow Model from the MTCO. In Section 4, by applying a comprehensive process design framework, we define the compliance rules between the CWM and the business process. Finally, Section 5 concludes the paper and gives suggestions for further research.

2 Related Research

Modeling of business contracts has been widely studied in the research community [1], [2], [3], [4], [5]. We have considered and enhanced those approaches when defining our contract ontology (MTCO, [6]). We have modeled obligation states to capture the information relevant to that obligation, as the fulfillment is being executed. Relationship between obligations, obligation states and performance events forms the foundation for the proposed CWM methodology. Our identification of the different obligation states is based on Yao-Hua Tan's work in [4]. The SweetDeal project [7] is closest to our MTCO and CWM in that they try to capture the semantics of different contract scenarios.

Many efforts have been made toward realization of electronic contracts using processes [5], [8], [9], [10]. Our methodology differs in two ways. First, in contract modeling, we focus on grasping the necessary semantic content irrespective of technology used. Second, in contract realization, we aim for integrating contract models with existing business processes rather than creating new process models. In that context our works is closest to van der Aalst's concept of workflow inheritance [11] used to compare public and private processes. In contrast to that work, our contract process model (CWM) comprises both public and private behavior; in addition, we compare the CWM with existing (private) processes using a less-formal method than van der Aalst, but more comprehensive.

3 Deducing the Contract Workflow Model (CWM) Using the Multi-tier Contract Ontology (MTCO)

In this section we first summarize our previous work on the contract knowledge modeling, in the form of a contract ontology named MTCO. Then we describe INCOTERMS delivery patterns, which are later used in our Sale of Goods case study for analyzing buyer's and seller's obligations. Based on the concepts of the MTCO, we define the Contract Workflow Model for the INCOTERMS-based case study.

3.1 MTCO

We follow Daskalopulu et al [1] in their identification of the roles of the contractual provisions. They prescribe certain behavior for the parties, under a set of conditions, and for a certain period of time. Thereby, contracts specify procedures that need to be followed. For example, the delivery terms inform the buyer regarding the time limit by which he has to inform the seller regarding his transporter, or any change in venue for the delivery, or delivery date etc.

We also follow [2] in their analysis of contracts from various perspectives such as:

A contract is an organized collection of concepts, obligations, permissions, entitlements, etc. A contract has also been viewed as a collection of protocols that specify its operational aspects (how the business exchange is to be carried out in reality) or simply parameters (the parties, product, price, delivery quantity, delivery date).

The MTCO is a combination of the above aspects as has been presented in [6]. A brief summary is presented below:

- The **Upper Level Core Contract Ontology** represents a general composition of a contract, which may be applicable across most of the prevalent types of contracts. The concepts defined here may be considered to be atomic blocks based on which all other contract types may be defined. Fundamental concepts like *role*, *consideration*, and *obligation* are defined here.
- The second layer is **Specific Domain Level Contract Ontology** or contract type specific collection of several contract type ontologies. Each ontology represents a specific contract type, such as Property Lease Rental, Employment Contract, Sale of Goods, etc. Each contract type inherits all fundamental features of the upper layer and thus specializes on the knowledge particular to that contract domain.
- The third layer, **Template Level Contract Ontology** consists of a collection of templates such as definitions for contract models, such as the International Chamber of Commerce's contract model for International Sale of Goods [12].

In our approach, we model the life-cycle of the contractual obligations as they pass through a set of states in response to identified *performance events*. For instance, an obligation, such as the seller's primary obligation to deliver, is *inactive* when the contract is signed. It becomes *active* when the buyer sends a purchase order and the seller accepts the order. Thereafter, the obligation becomes *fulfillment triggered* when the seller packs and starts the delivery process. Once the buyer accepts and sends notification of acceptance to the seller, the seller's obligation to deliver may become *fulfilled*. On the chance that the delivery is rejected and compensation is sought by the

buyer, then the seller's primary obligation is *pending* while the reconciliatory obligation is now *active*. In case the buyer cancels the order then the seller's obligation to deliver becomes *cancelled*. This proposed classification of obligation states, is the key concept for creating process-based contract choreography in the form of the Contract Workflow Model.

In the next section we discuss the INCOTERMS delivery patterns, which are later used in our Sale of Goods case study for analyzing the obligations and the expected business behavior from the seller and the buyer respectively.

3.2 Case Study Scenario: INCOTERMS Delivery Patterns

Sale of Goods business contracts form the domain of interest for our knowledge models. Such legal contracts, pertaining to purchase and sale, cover a wide arena. We chose a smaller domain of interest as a case study, to model our research goals on. We chose to analyze and model the contractual knowledge from a recommended standard of contract patterns ICC's INCOTERMS [13]. INCOTERMS provide a set of international rules for the interpretation of the most commonly used trade terms in foreign trade.

INCOTERMS form only a part of the vast contractual knowledge contained in a Sale of Goods contract, as recommended by ICC International Sale of Commercial Goods. These are delivery terms agreed between the two parties involved in a sale of goods business transaction. They are internationally accepted as standardized patterns for delivery terms. In this paper, we propose the use of INCOTERMS *as a horizontal extension or an individual reusable ontology in the specific domain level of the MTCO*. Either the "delivery terms" of a Sale of Goods contract may refer to one of the INCOTERMS standard codes, or the contracting parties may define their own, as the established business practices.

The main INCOTERMS concepts are summarized below:

- **Actor.** Two or more parties who participate in the purchase and sale of goods.
- **Obligation.** An obligation is a promise or commitment made by one party to the other, which is backed up by legal intent to fulfill and uphold the promise.
- **Role.** The part an actor plays in the specific contract (i.e. seller, buyer, etc.).
- **Goods.** Goods are legally defined as commodities or items of all types, expecting services, which are involved in trade or commerce. Goods are characterized by a description, technical specification, type of packaging required, type of cargo etc.
- **Performance.** Every obligation is expected to be fulfilled through the execution of some business, economic or legal action. This expected business behavior is termed as *performance*.
- **Delivery Terms.** Delivery terms are agreed terms and conditions, which govern the actual process of delivery of goods between the seller and the buyer.
- **Packaging.** In most cases the parties know beforehand which type of packaging is required for the safe carriage of the goods to the destination. As the seller's obligation to pack the goods may vary depending upon the type and duration of the transport, it has been stipulated that the seller is obliged to pack the goods in the way it is required for the transport.
- **Inspection of Goods.** In many cases the buyer may be advised to arrange for the inspection of goods before, or at the time the goods are handed over by the seller.

- **Payment Terms.** In relation with delivery terms, Sale of Goods includes payment terms, which are expected in exchange for the goods delivered. Though the INCOTERMS does not go into specific details for the payment terms, the obligation and the performances are outlined.

In order to specify the obligations, roles and responsibilities of different delivery protocols, the INCOTERMS have been grouped in four different categories. In our study we will use the term Ex Works, whereby the seller makes the goods available to the Buyer at the seller's own premises, to bring the goods to the country of destination. In Table 1 below, we present a summary Ex Works, in order to elucidate the nature and type of obligations that are present in any such delivery term. In the next section, we base a sample contract example (provided in Annexure A) on the Ex Works obligations from Table 1, and thereafter, we deduce a process-based contract choreography in the form of the Contract Workflow Model.

Table 1. Extract of distribution of obligations in Ex Works delivery terms

Obligation	Seller	Buyer
Provision of goods and Comm. Invoice	Yes	No
Delivery	Place the goods at the named place of delivery on the date or within the period stipulated.	Take delivery as soon as they have been placed at his disposal.
Payment of price	None	Pay the price as provided in the contract.
Notice to the buyer	Give notice as to when and where the goods shall be placed at the buyer's disposal.	Not applicable
Packaging, marking	Provide at his own cost packaging as it is required by the circumstances relating to the mode of transport, destination etc. Packaging is to be appropriately marked.	Providing the transport modalities and informing the seller of the required packaging.
Inspection of goods	None	Pay the shipment inspection and inspection mandated by the country of exportation.

3.3 Contract Workflow Model for the INCOTERMS Case Study

As we have explained in Section 3.1, the Multi Tier Contract Ontology defines the states the contract obligations may pass, and based on them, a set of performance events that initiate transitions of those states. The expected flow of the performance events (i.e. business actions [14]) is then modeled as a *Contract Workflow Model*. In [15] we have defined the CWM as:

A partial choreography of performance events for each of the parties concerned as deduced from the perspective of the governing business contract and is an indicative model for the contract compliant business process model.

Following this, below we outline the procedure for obtaining a CWM from a contract:

- The contract document is analyzed to extract the data contained in it.
- The extracted data are compared and matched to the specified meta-data concepts as established in the Multi Tier Contract Ontology. The two steps are repeated iteratively to obtain all the concepts and their instances in the contract.
- From the identified contract type ontology (in this study – INCOTERMS), the common obligations and their performances, defined or suggested, are considered. Added to the specific information from the contract instance, the list of all stipulated obligations and the required or inferred performance events is obtained.

By applying the outlined procedure to our Sale of Goods contract instance (provided in Annexure A), which is based on the Ex Works/INCOTERMS, we sort out the list of the contract obligations and associated performance events (Table 2):

Table 2. Partial list of obligations and deduced performance events from the contract sample

Obligation	Possible list of Performance Events
Obligation to deliver	Deliver goods (implied from Ex Works)
	Take order (implied from the contract)
	Arrange carrier (implied from the contract)
	Deliver at the agreed place for point of destination (implied from Ex Works)
	Notify buyer on delivery (implied from the contract)
	Compensate rejected delivery (implied from the contract)
Obligation to package	Pack goods (implied from Ex Works)
	Mark goods (implied from Ex Works)
Obligation to Pay	Inspect Goods (implied from Ex Works)
	Notify seller of any discrepancy and demand remedy (implied from the contract)
	Pay for goods (implied from Ex Works)

The identified obligations and the performance events are further grouped according to the actor performing them. In our case study, the information on responsibilities for the obligations is identified from the list of distribution of obligations in Ex Works delivery terms (Table 1) and the contract example (Annexure A).

As we explained in the previous section, the contractual obligations pass through a set of states in response to the identified performance events. Following this, the performance events from Table 2 are ordered in a time-based sequence according to the life-cycles of the identified obligations. For instance, the seller's obligation to deliver is changed from the inactive state (when the contract is signed) to the active state when the seller takes (receives) the order from the Buyer; the obligation changes to the fulfillment-triggered state when the goods are to be packed and when the carrier is arranged. By continuing in this way, all identified performance events are ordered in a common, process-based, contract choreography.

In order to obtain a “valid” process model, the obtained choreography is then transformed to a Business Process Modeling Notation model [17]. The BPMN is an expressive, graphical process modeling notation, easy understandable by process modelers. In [16], we have defined the core mapping patterns between the concepts of the CWM and the BPMN:

- The performance event to the BPMN sub-process,
- The obligation states to the BPMN events,
- The choreography of performance events to the BPMN sequence flow,
- The simultaneous processing of two or more performance events to the BPMN AND gateway,
- The exclusive processing of performance events to the BPMN XOR gateway, etc.

Following the patterns, we have deduced the CWM for our Sale of Goods example to the BPMN form (Figure 1):

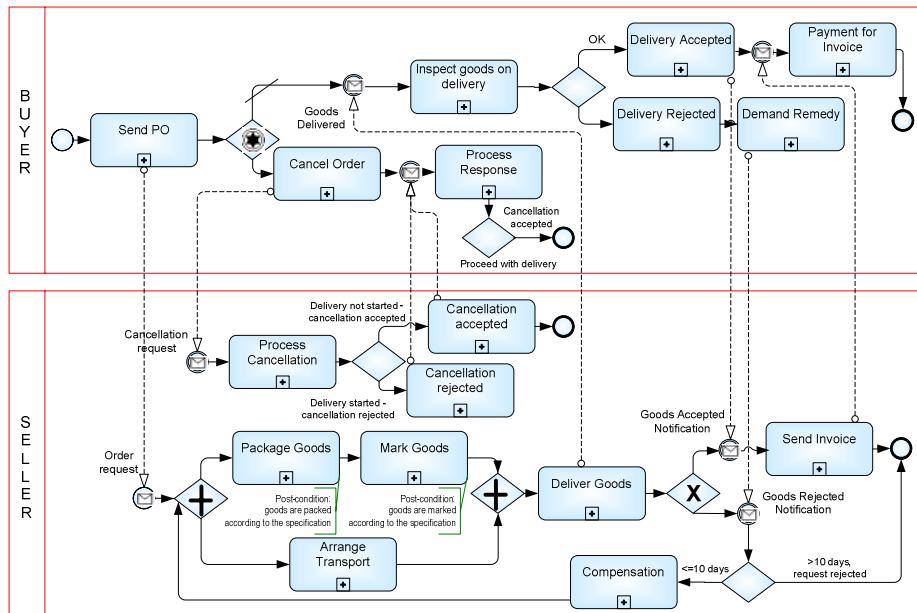


Fig. 1. The Contract Workflow Model of the buyer and the seller represented as a BPMN high-level process

The CWM in Figure 1 is, by the previously described procedure, partitioned on a per-actor basis. The performance events from Table 2 are presented as sub-processes, which are triggered by obligation state changes (represented as message-based events in Figure 1). In this way formalized, the CWM forms a high-level business process model, which complies with the agreed contractual terms and conditions. In such way conceptualized, the CWM is used for comparison with existing business processes.

4 Compliance Between CWM and Existing Business Processes

In the B2B context, the CWM specifies the agreed contract patterns in the form of a process. As defined the previous section, the CWM represents both the interaction and coordination flow. In terms of interaction, the CWM specifies the protocol for the public message exchange. In terms of coordination, the CWM specifies the flow of business activities. A CWM differs from a business process in the way that it defines the flow of high-level activities. For that reason, the CWM of the seller presented as a BPMN model in Figure 1, may be considered as a partial and high-level specification of seller's business process.

As shown in the previous section, the BPMN technique allows for visualized process modeling. A business process, designed as a BPMN model may be further converted to a BPEL4WS specification [18], using a set of mapping formalisms. In contrast to the language format of a BPEL4WS process specification, a BPMN model enables designer to intuitively and easily capture various aspects of process design.

In the following, we define a process design framework to provide a basis for a systematical examination of compliance between a CWM and a business process, both given in the form of BPMN models. By following the framework, we identify *compliance rules* when mapping CWM to business processes. The total set of rules covers the transformations that may be applied during the assessment of compliance between a CWM and a business process.

4.1 Process Design Framework

In this section we introduce a conceptual framework to classify different aspects that constitute design of a BPMN process. The framework is based on the modelling aspects of workflows, as proposed in [19], [20].

The *functional* aspect considers the activities of a process. Functionality of each activity is determined by three elements: the activity name which describes the result to be fulfilled, exchanged messages, and input and output constraints, i.e. pre-conditions and post-conditions. In the BPMN, activities can be non-atomic, such as *sub-processes*, or atomic, such as *tasks*.

The *behavioural* aspect depicts the control flow of a process. For specification of dependencies and coordination rules among activities, process models rely on a set of basic control flow constructs: sequence, parallel execution (AND), conditional branching (OR/XOR) and looping. In addition, activities may be triggered by events, which are signalled by internal or external occurrences. In the BPMN semantics, *the normal flow* is used to model the basic sequence, while *gateways* are used to model AND, OR/XOR controls, and loops. Activities might be triggered by three types of events: start, end or intermediate. The BPMN includes the events that affect the sequence or timing of activities of a process. Events are categorized as Start, End, or Intermediate (may be time-, message, cancel-, rule- or error-based).

The *informational* aspect of concerns the information concepts needed for representing process instance data. In a process specification, instance data are represented using internal process attributes upon which flow rules are set and controlled, and with the information that the process exchanges with the external environment in the form of messages (and documents). In the BPMN, the internal data are stored as the

properties of a process, *message flows* indicate information to be exchanged between two entities, while *data objects* are used to depict documents attached to messages without any information on their content and structure.

The *organizational* aspect concerns the distribution of responsibility for executing activities of a process. This responsibility is assigned to business roles such as seller or buyer. By using roles it is possible to dedicate and control responsibilities of participants engaged in a process. The BPMN uses *pools* to represent process participants (i.e. business roles).

The *transactional aspect* manages consistent execution of a set of activities. As process activities may have short or long duration, process transactions comply with two different models. The *atomic transaction model* [21] governs shorter activities, that all agree to enforce a single visible result by two-phase commit. The *long-running transaction model* [22] administers more durable activities, where each activity imposes a globally visible outcome independently of the others; when an activity fails, the parent transaction rolls back to a consistent process state by compensating activities that had successfully completed. In the BPMN, a *transaction* is one mean to ensure the consistent process states by forcing all activities involved to complete or to be cancelled by following one of the two transactional models. Another way is to define custom error-handling that will lead the process to specified consistent states.

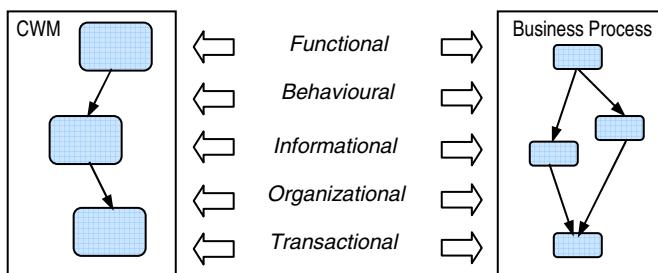


Fig. 2. The five-aspect framework used for examination of compliance between CWM and business processes

By following the described framework, in the next section we define rules for compliance between CWM and business processes, for each process design aspect.

4.2 Compliance Rules Between CWM and Business Processes

To ensure contract obligations are to be fulfilled in the execution phase, the CWM and the business process need to be compliant. This means that, when enacting a contract, for a business process to be compliant with a CWM, the process must trace the states of the CWM. That ensures that from the CWM view, all obligation states might be monitored. As described in the previous section, five aspects constitute design of a BPMN process; this means that every state in the process comprises the current status of each of the design aspects. To support a CWM, a business process must be thus compliant with the CWM in all five design aspects.

Functional Aspect. The CWM models functionality at the level of sub-processes, while the business process does it on the task level. When comparing the CWM with the business process from the functional perspective, we investigate whether activities in the business process are compliant with the CWM activities, with respect to their results, message exchange, and/or imposed pre- and post-conditions. Following this, the functional aspect of a business process will be compliant with a CWM, if the design of the business process satisfies the following rules:

- *Result/Messages.* One or more activities in the business process correspond to one activity in the CWM, where the process activities jointly provide the same result and exchange the same messages as the CWM activity. This ensures that the business process can trace the messages and the results of activities in the CWM.
- *Constraints.* The pre-conditions of the activities in the business process must be same or weaker than in the CWM, while post-conditions must be same or stronger. This ensures that the constraints of the activities, as defined in the CWM, are supported in the business process.

To illustrate compliance of the functional aspect, we consider the CWM of the seller in the example from Figure 1, and an excerpt of the seller's business process (due to space limitations we cannot depict the whole business process). The contract implies (Figure 3a) that goods are to be packed by executing the activity “Package goods” with the post-condition that packaging must be done according to a given specification. In the business process, packaging of goods is designed as shown in Figure 3b.

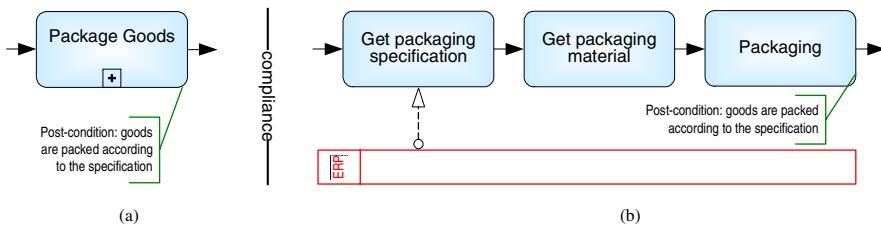


Fig. 3. Packaging of goods – in the CWM (a) and in the business process (b). The models are compliant by the result/message and the constraints rules.

In this case, the result/messages rule is satisfied, because the joint result of the three activities in the business process corresponds to the result of the CWM activity and the message exchange is equivalent (since none of the business process activities exchange the external messages). The constraints rule is also satisfied, because the post-condition of the last activity in the business process is equivalent to the post-condition of the activity in the CWM.

Behavioural Aspect. When investigating the behavioural aspect, the control flow in the business process must be compliant with the flow of activities in the CWM. It means that the business process must be able to trace all the control flow states from the CWM. This implies that the behaviour of a business process will be compliant with the behaviour of a CWM, if the business process fulfils the following rules:

- *Ordering.* In the business process, ordering of activities must be the same or stronger than in the CWM. This means, for instance, that the parallel flow in a CWM is compliant with both parallel and sequence flow in a corresponding business process.
- *Branching.* In the business process, branching must be designed such that every branching condition in the CWM has a corresponding branching condition in the business process. This means the business process may also specify additional conditional branches.
- *Events.* Events, as specified in the CWM, must exist in the business process. An exception of this rule is mapping of message-based events defined in a CWM, which denote interaction points between the partners; those events may be modelled in a corresponding business process either as events or as “receive” tasks, because both elements model an equivalent behaviour from the BPMN view.

An example of the event and branching rules is given in Figure 4. In the seller’s CWM, the event that captures the buyer’s cancellation notification (Figure 4a), and the activity “Receive cancellation request” in the business process (Figure 4b), model the equivalent actions. As for the branching rule, the CWM includes two branches to model possible flows upon receiving a request for cancellation, while the business process includes an additional branch that examines the number of days passed from the receipt of order. The compliance between the CWM and the business process holds, because in the business process, the value for the number of days may be set high enough so that the task “Send request for penalty” is never triggered.

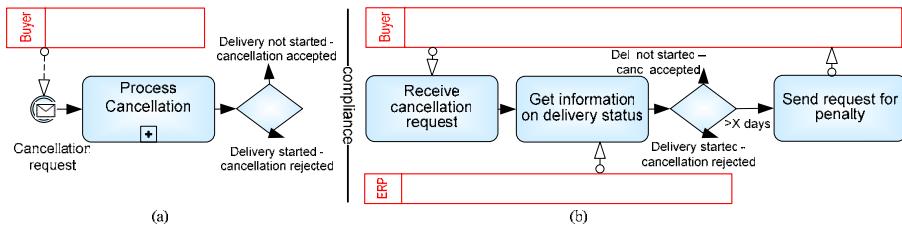


Fig. 4. Processing order cancellation - in the CWM (a) and in the business process (b). The models are compliant by the event and the branching rules.

Informational Aspect. Regarding the internal process data, the CWM includes the information upon which contract flow rules are controlled (such as buyer’s address, timeframes for order cancellation, etc.). Regarding data exchanged with the external environment, as explained at the end of Section 3.3, the CWM indicates only the messages that are to be exchanged (and not contained documents). Following this, the informational aspect of a business process will be compliant with a CWM, if the business process satisfies the following rule:

- *Information concept.* The business process must include at least the information concepts included in the CWM. This means the business process must provide required messages, and support required internal process data. The business process may support additional informal concepts not required by the CWM, in the form of messages or internal process attributes.

As an illustration, in the example in Figure 1, the CWM specifies a timed-dependent condition expression (10 days) for the notification of the reject of goods, where the number of days is hold as the internal data. In Figure 5, it may be seen that the business process supports the required time attribute, which means that the rule for the inclusion of the information concept is satisfied.

Transactional Aspect. By following the contract terms, a CWM specifies what are to be considered the consistent process states. For instance, a customer may agree with a travel agency that it is obliged to confirm booking of both a flight and a hotel, or if this is not possible, to notify the customer on inability to perform the bookings. The consistent states of the corresponding CWM are, therefore, those when both bookings are done, or none of them. This implies that a business process, in order to be compliant in the transactional aspect with a CWM, must satisfy the following rule:

- *Consistent states.* The business process must support either an adequate transactional model (i.e. atomic or long-running) or a “custom” error-handling flow of activities which lead to the same consistent process states as defined in the CWM.

For instance (Figure 1), when the seller receives the “Goods rejected notification” the CWM specifies a compensation sub-process to be started (if the notification is sent less than 10 days after delivery). In the business process (Figure 5), the compensation procedure is implemented as a long-running transaction. However, as the consistent states are equivalent (i.e. both processes are revised to the state before packaging of goods), it means the business process is complaint with the CWM.

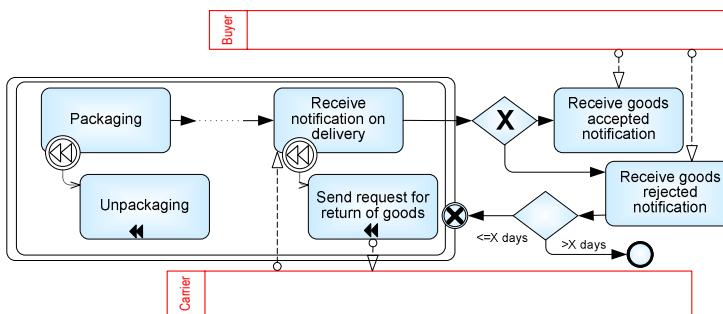


Fig. 5. Reject of goods in the business process; the procedure is compliant with the reject of goods in the CWM in Figure 1, by information concept and the consistent states rules

Organizational Aspect. In the CWM, the organizational aspect is represented at the level of the contracting parties, while in the business process activities are associated with the parties that actually perform those activities. The organizational aspect as defined in a business process will, therefore, be compliant with a CWM, if the following rule is fulfilled:

- *Role.* The activities in the business process must be under supervision of the party that is responsible for the corresponding business activities in the CWM.

In the example in Figure 1, it is the seller who is responsible (on the buyer’s request, as specified in the contract sample in Annexure A), to deliver the goods to a point specified

in the contract (Figure 6a). In the business process, as we see it in Figure 6b, the seller contacts a transport company (the carrier) to arrange the delivery. However, as the activities of the carrier are under the supervision of the seller’s process, from the buyer perspective, the contract terms are fulfilled. In contrast, if the seller process might not support management of the delivery, such as when the seller cannot himself provide the delivery, or when it cannot supervise the delivery (as there is no communication between the seller and the carrier), the contract terms cannot be fulfilled.

4.3 Compliance Assessment

The transformation rules stated in the previous section give the possibility to assess the compliance between a CWM and a business process. The assessment requires examination of all five design aspects in order to discern whether a business process conforms to a CWM. In the following, we describe the main steps for assessing the compliance between a CWM and a business process:

- Examining the **functional aspect**, the activities in the CWM are mapped to activities in the business process, by following the result/messages rule. This means that a sub-process from the CWM will be mapped to one or more tasks in the business process. At the end of the procedure, non-mapped CWM sub-processes, if such exist, are denoted. The pre- and post-conditions of every mapped sub-processes in the CWM are further controlled against the conditions of the tasks in the business process, by following the constraints rule. Those sub-processes which do not comply with the rule are denoted.
- The **transactional aspect** is examined by locating the transactions and/or error-handling sub-processes in the CWM. By following the consistent states rule described in the previous section, the located elements in the CWM are compared for the compliance with the business process. If they do not lead to the same consistent process states, the CWM elements are denoted.
- For the **behavioural aspect**, first the business process tasks that are mapped by the CWM sub-processes in two previous phases are enclosed in those sub-processes. The rest of the tasks in the business process are, if possible, “hidden” by being labelled as *silent* (i.e. not considered in the process flow [23]):
 - a. All tasks that model internal business behaviour (such as “verify order”, “make goods”, etc.) could be denoted as silent, as they are neither regarded by the contract nor by the public interactions.
 - b. The tasks that model public interaction (such as “send order acknowledge to the buyer”) are hidden from the process behaviour if they conform to the protocol inheritance notion [11].

As an illustration of the hiding principle we consider the example in Figure 6. The business process task “Send notification on shipment” models an interaction not specified by the contract. However, by following the notion of the protocol inheritance, the execution of a task in the sequence flow may be hidden by disregarding it (in the contrast, it would not be possible to hide a task if it follows an implicit XOR gateway, as it might prevent the execution of the other task in the gateway).

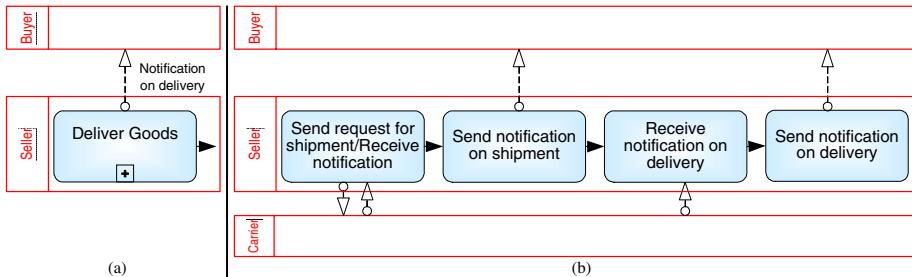


Fig. 6. Deliver goods in the CWM (a); hiding redundant activities in the business process (b)

The compliance of events and gateways between the CWM and the business process is examined by applying the event and branching rules. Those elements that are coupled are labelled equally.

If after the described steps all tasks are hidden, and all events and gateways are labelled, the CWM and the business process are compared for the flow equivalence using the *branching bisimulation*¹ algorithm; if, not the conflicting tasks, events and/or gateways are denoted.

- The realisation of **informational aspect** is examined by applying the information concept rule. The compliance of the messages is examined when the result/messages rule of the functional aspect is investigated. The internal CWM information concepts are compared for matching with those in the business process, by comparing the BPMN property elements. If some information concept from the CWM is not included in the business process, it is denoted.
- For the **organisational aspect**, it is examined if the roles (represented with pools) that supervise the compliant sub-processes in the CWM and in the business process are same, by following the role rule. The CWM roles that might not be matched are denoted.

The described procedure has the purpose to guide the process designer through the assessment of compliance between a CWM and a business process. During the assessment, each of the design aspects is examined. If all design aspects are compliant, it means the CWM may be realised with the existing business process. If not, the non-matched CWM elements are used as a guideline on how the business process should be changed. This means that the proposed methodology may be used in the way that the found non-compliances form a knowledge base of the elements and logic required for the process evolution (such as which activities must be added or removed; in what way the control flow should be changed; what information concepts are missing, etc.). This is especially important for tracing the “un-happy” CWM paths (i.e. exceptions, cancellations, etc.), as in the most of cases those are not supported by existing business processes due to lack of knowledge.

¹ The relation of *branching bisimilarity* may be used to compare observable behavior of two state-based process models, as it allows abstracting from the non-observable behavior, i.e. silent transitions. For more details, the reader is referred to [24] and [23].

5 Conclusion and Future Work

In this paper, we have proposed a methodology for transforming contract requirements to concepts of processes and comparing them for compliance with existing executable business process models. Our main aim was to define an approach for bridging the contract and process domains in a comprehensive and realistic manner. To achieve that, in the contract domain, we aimed for concepts that might grasp broad contract options; in the process domain, we strived for comprising the aspects that constitute process design. We believe that our methodology, with its concepts, subsumes the following issues:

- **Mapping of contract requirements to process concepts.** The utilization of the Multi Tier Contract Ontology as a framework for modeling contract knowledge enables identification of the states through which contract obligations are passing. By composing those states in a time-based sequence, the contract choreography (namely, the Contract Workflow Model) is obtained in the form of a high-level BPMN process model.
- **Traceability of contract requirements.** By using the framework based on five main aspects that constitute process design, a set of compliance rules between the Contract Workflow Model and low-level (i.e. executable) business process models is defined. Those rules are used to, on the per-aspect basis, examine compliance of an existing business process with the CWM. In this way, traceability between contract terms and conditions, and existing business processes is enabled.
- **Process interoperability.** By assessing compliance of the partners' business processes with the corresponding Contract Workflow Models, it is determined whether those processes may realize the required contract requirements, i.e. whether they are interoperable on the level of the rules implied from the contract. In case a process is not compliant with a respective CWM, the results from the assessment might be used as a guideline for needed changes of the process, for each of five design aspects.

A subject for further work is contract monitoring. Though we have not focused on that aspect in this paper, the identified mappings between the CWM and the business process can be used for monitoring the commitment fulfillment and performance analysis after each contract execution. Another subject for further research is to, by considering the suggested set of compliance rules, conceptualize the used process design framework in the form of a process ontology, and as such, employ it as a basis when comparing the CWM with the existing business process.

References

1. Daskalopulu, A., Sergot, M.: The Representation of Legal Contracts. *AI and Society* Vol. 11, Nr 1&2. Springer-Verlag (1997)
2. Griffel, M., et al.: Electronic Contracting with COSMOS - How to Establish, Negotiate and Execute Electronic Contracts on the Internet. Proceedings of the Int. Workshop EDOC '98, San Diego 1998
3. Lee, R.: Toward Open Electronic Contracting. *The International Journal of Electronic Markets*, Vol. 8, Nr 3 (1998)

4. Tan, Y. H., Thoen, W.: A Logical Model of Directed Obligations and Permissions to Support Electronic Contracting. *The International Journal of Electronic Markets*, Vol. 10, Nr 1 (2000)
5. Karlapalem, K., Dani, A., Krishna, R.: A Framework for Modeling Electronic Contracts; International Conference on E-R Modeling (ER 2001). LNCS 2224 pp 193 – 207
6. Kabilan, V. Johannesson, P. Rugaimukammu, D. Business Contract Obligation Monitoring through use of Multi-Tier Contract Ontology. Proceedings of Workshop on Regulatory Ontologies (Worm CoRe 2003), November 2003, Italy. Springer-Verlag 2003 LNCS 2889, pp 690-702
7. Grosof B., Poon T.: SweetDeal: Representing Agent Contracts with exception using XML rules, Ontologies and process descriptions. Proceedings of the 12th International World Wide Web Conference (WWW2003), Budapest, Hungary. ACM (2003)
8. Alonso, G., et al.: WISE: Business-to-Busibness E-Commerce. Proceedings of 9th International Workshop on Research Issues and Data Engineering, Sidney, Australia. IEEE Computer Society (1999)
9. Grefen, P. et al.: CrossFlow: Cross-organizational Workflow Management in Dynamic Virtual Enterprises. *International Journal of Computer Systems*, Vol. 15., No. 5, (2000)
10. Angelov, S., Grefen, P.: Support for B2B E-Contracting – The Process Perspective. 5th Int. Conf. on Information Technology for Balanced Automation Systems in Manufacturing and Services (BASYS'02), Mexico. IFIP Conference Proceedings 229. Kluwer (2002)
11. Aalst, W. M. P. van der: Inheritance of Interorganizational Workflows to Enable Business-to-Business E-commerce. *Electronic Commerce Research* Vol. 2, Nr 3. Springer-Verlag (2002)
12. ICC International contract for sale of goods, published by ICC books, 2002
13. Ramberg, J.: ICC Guide to INCOTERMS 2000. Understanding and Practical Use; International Chamber of Commerce (2000)
14. IEEE's Suggested Upper Merged Ontology, <http://suo.ieee.org>
15. Kabilan V, Zdravkovic J, Johannesson P. Use of Multi-Tier Contract Ontology to deduce Contract Workflow Models for Enterprise Interoperability. Proceedings of 2nd INTEROP-EMOI open workshop on Enterprise Models and Interoperability collocated with CAiSE 2005, Porto
16. Kabilan V. Contract Workflow Model Patterns Using BPMN. Proceedings of the 10th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 05), co located with Caise 2005, Porto.
17. White, S.: Business Process Modeling Notation, version 1.0, Business Management Initiative, May 2004; <http://www.bpmi.org>
18. BEA, IBM, Microsoft, SAP and Siebel. Business Process Execution Language for Web Services (BPEL). <http://www-106.ibm.com/developerworks/library/ws-bpel/>, June 9 2004.
19. Jablonski, S. A Software Architecture for Workflow Management Systems. Proceedings of the Ninth International Workshop on Database and Expert Systems Applications (DEXA'98) (Vienna, Austria, August 1998). IEEE Computer Society, 1998, 739-744
20. Rausch-Scott, S. TriGSflow – Workflow Management Based on Active Object-Oriented Database Systems and Extended Transaction Mechanisms. PhD Thesis, Univ. at Linz, 1997
21. Bernstein, P., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987
22. Garcia-Molina, H. Modeling Long-Running Activities as Nested Sagas. *IEEE Data Engineering Bulletin*, 14, 1, 1991, 14–18

23. Aalst, W. M. P. van der, Basten, T.: Inheritance in Workflows. An Approach to Tackling Problems Related to Change. In: Theoretical Computer Science, Vol. 270(1-2). (2002) 125-203
24. Groote J. F., Vaandrager F.: An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence. In: Proceedings 17th ICALP. Lecture Notes in Computer Science Vol. 443. Springer-Verlag, (1990) 626-638

ANNEXURE A: Sample Sale Contract Model

CONTRACT FOR SALE OF GOODS

Agreement made and entered into this [12th Jan 2005], by and between [ABC Computers Incorporated Ltd], of [Österrogatan 17, Kista, Sweden], herein referred to as "Seller", and [Financial Movers AB], of [Strandvägen 2, Stockholm, Sweden], herein referred to as "Buyer".

1. Seller hereby agrees to transfer and deliver to buyer, within 30 days from the date of order receipt, the following goods: DELL PC Dimension 4550, Intel Pentium 4 processor, 333 MHz DDR SDRAM desktops conforming to the technical specifications. Product details specified separately.
2. Buyer agrees to accept the goods and pay for them in accordance with the terms of the contract.
3. Buyer and Seller agree that identification shall not be deemed to have been made until both parties have agreed that the goods in question are to be appropriated and fulfil the requirements of performance of said contract with the buyer.
4. Delivery shall be in accordance to standard INCOTERMS "EX-Works". However, upon buyer's request, the seller is obliged to make the necessary arrangements for transportation and all costs pertaining to that shall be borne by the buyer.
5. Buyer agrees to pay for the goods at the time they are delivered and at the place where he receives said goods.
6. Goods shall be deemed received by buyer when delivered to address of buyer as herein described.
7. Until such time as said buyer has received goods, all risk of loss from any casualty to said goods shall be on seller.
8. Seller warrants that the goods are free from any security interest or encumbrance, that they shall be free from it at the time of delivery, and that he neither knows nor has reason to know of any outstanding title or claim of title hostile to his rights in the goods.
9. Buyer has the right to examine the goods on arrival and has 10 days to notify seller of any claim for damages on account of the condition or quality of the goods. His failure to either notice seller or to set forth specifically the basis of his claim will constitute irrevocable acceptance of the goods.
10. This agreement has been executed in duplicate, whereby both Buyer and Seller have retained one copy each, on [12th Jan 2005].

Resource-Centric Worklist Visualisation

Ross Brown¹ and Hye-young Paik²

¹ Faculty of Information Technology,
Queensland University of Technology, Brisbane, Australia
² School of Computer Science and Engineering,
University of New South Wales, Sydney, Australia
`r.brown@qut.edu.au, hpaik@cse.unsw.edu.au`

Abstract. Business process management, and in particular workflow management, are a major area of ICT research. At present no coherent approach has been developed to address the problem of workflow visualisation to aid workers in the process of task prioritisation. In this paper we describe the development of a new, coherent approach to worklist visualisation, via analysis and development of a resource-centric view of the worklist information. We then derive appropriate visualisations for worklists and the relevant resources to aid worker in decision making. A worklist visualisation system has been implemented as an extension to an open-source workflow system, YAWL (Yet Another Workflow Language).

1 Introduction

Visualisation techniques offer powerful tools for understanding data and processes within complex systems. However, visualisation in the area of Business Process Management (BPM), and in particular workflow management systems lags behind the state of the art in other areas such as medicine, engineering and mining [1]. A recent Gartner's report suggests that many business organisations consider BPM to be a fundamental driver of business innovation. This is demonstrated by the large amount of money and expert resources invested in business process modelling, analysis and roll-out of the models. Workflow Management Systems (WfMS) play a vital role in BPM in that the business process models are implemented and executed through a WfMS, which routes and dispatches the tasks defined in a model to the individual workers¹. The result of "routing" tasks is presented to the workers as a *worklist*. A worklist can be understood as a "to-do" list of tasks that the workers need to carry out in order to complete the process defined by the model.

The success of business process models depends on communicating them to the model consumers effectively. However, modern workflow systems have largely overlooked the needs of the workers of understanding their given tasks in the manner that would help manage them efficiently. For example, it is quite common that the workers would have questions such as "how urgent is this

¹ The workers are the "consumers" of the model who will carry out the tasks. In this paper, we use the terms "workers" and "model consumers" interchangeably.

task?”, “who else can do the task?”, “where do you have to go to carry out the task? (eg., where is this meeting room B809)”, “do I have enough resources? (eg., are there enough chairs for 20 people in the meeting room B809)”, etc.

A typical representation of a worklist includes a list of tasks with short textual descriptions, and/or attachments (eg., email, document forms, etc). It, however, does not include any support (context) information about the tasks that may assist the worker in planning the tasks. At any point in time, a given worker may be involved in many workflows and may thus be presented with a large to-do list. The worker needs to have available tools to help them decide which would be the “best” task to undertake next.

We believe that visualisation techniques can be applied to many areas of BPM due to their previous use in application domains that support decision making processes. In decision support systems, information is typically provided to enable the user to be adequately informed to the direction to be taken for a particular scenario. This applies to all levels of business systems, and to BPM as a whole.

For the purpose of this paper, we limit the scope of the work to the area of workflow management, in particular, managing worklists. We apply a visualisation technique to provide workers with information about the context of a task, in order to improve their understanding of the process models and the communication of such models between the model designers and the consumers. The visual information is designed to help workers make decisions in managing worklists (eg., accepting, postponing, delegating, or rejecting tasks).

In this paper, we propose a generic visualisation framework that is used to provide support (context) information about the tasks in a worklist. Our contributions are three folds:

- An analysis of the decision making process in managing workflow tasks, especially in relation to the resources available to the worker
- A novel and generic visualisation technique for worklists
- The implementation of the framework as a proof of concept

The rest of the paper is organised as follows: Section 2 investigates the state of the art in worklist visualisation. Section 3 details the development of a resource centric approach to the management of worklists. Section 4 explains the mapping of important worklist resources to appropriate visualisation techniques to aid the process of task selection by workers. Section 5 describes the general approach to using these visualisations in workflow systems. Section 6 details the implementation of a visualisation system incorporated into a workflow system. The paper then concludes with a discussion of future work.

2 Related Work

Computerised data visualisation is a broad field that has its beginning in pictorial representations used in pre information technology times [2]. Today it has developed to the point of being one of the main areas of application for computer graphics, and is a powerful tool for the analysis and presentation of data [1]. Many

areas in science and mathematics have benefited from the exploitation of modern computing power in data exploration. Business experts are now using advanced computerised visualisations to analyse multi-dimensional business data and processes. We believe that there is a very good opportunity to apply these leading edge techniques to the visualisation of business processes models and their execution.

The present state of play for the visualisation of BPM systems uses process state tables [3], simple icon based 2D visualisations [4] and more complex 2D visualisations [5]. Some have explored the use of 3D extensions to 2D diagrams [6] to 3D representations using such techniques as Cone Trees [7] to full virtual reality implementations for distributed interaction with detailed process models [8]. Some used abstract representations such as Self-Organising Maps (SOM) [9].

A body of research has been carried out into visualisation of business process data and is collectively known as BizViz (Business Visualisation) [10,11,12]. Bizviz consists of the visualisation of data alone, and not business process information. At present, it is ad hoc in nature, without a rigorous assessment of a number of the following factors: potential valid visualisation techniques from other fields and business requirements for such visualisations.

While there has been evidence of research into user requirements for business process modelling [13,5], much work still remains with regards to the following:

- Data gathering for requirements analysis, the current research is often tied to software implementations which restrict creative solutions;
- No real evidence of systematic analysis of sophisticated 2D and 3D visualisation techniques for use in complex business process models;
- Abstract representational techniques are often ignored despite their power in representing multi-dimensional data that occurs in business systems;
- Application domain information is not factored into the representations;
- No assessment of visualisation effectiveness via real case studies.

What is needed is a thorough data gathering-based analysis of user requirements for the visualisation of business processes, and the analysis of the many 2D/3D techniques and visualisation wisdom for such representations. In particular, the area of concurrent process visualisation [14] is expected to provide many useful visualisation techniques. Furthermore, there is a need to provide an approach to visualisation of business processes that accounts for domain specific factors in their representations. Such a visualisation approach needs to allow for both the designers [15] and the users of the business process model [3,13], as both these people have different requirements for visualisations, with regards to design, analysis, and usage tasks.

What these other workflow visualisation techniques lack is a focus on supporting information to assist the worker in managing the tasks in a worklist. Each of the techniques provides a presentation of the worklist that is rudimentary in nature, lacking any support information for the main task required by a workflow system; deciding to accept, delegate, suspend a presented task. We believe this should be the main reason for such workflow visualisations, and that an analysis of this choice process and derivation of appropriate visualisation techniques is required to support this process. Analysis of such requirement is best taken from

a resource oriented point of view [16], as the available resources in an organisation control the acceptance or rejection of the task² into the active worklist of the worker. We now proceed to analyse this worklist management problem from a resource perspective, in order to derive appropriate worklist visualisations.

3 Resource-Centric Views of Worklists

In this section, we introduce a notion of resource-centric views for worklists. It should be noted that we use the term resources specifically to refer to any work environment element or context that may be considered when workers make decisions in managing their tasks.

3.1 Example Scenarios

To illustrate our concept, we use the following two simple workflows as running examples throughout the paper³. The first case represents a process given to a university student who needs to obtain proper access to the university facilities. According to Fig. 1, the student will have to visit several service centres situated in different locations in the campus to obtain a computer account and a student card. The second case describes a stocktaking process given to an asset management officer who has to record all computer assets managed by a company. Fig. 2 describes that, after stocktaking is announced, the officer has to plan and schedule field trips to various sites to physically locate an asset and record the asset number using a barcode scanner. This process will continue until all the sites have been visited.

3.2 Analysis of Resources for Worklists

For the workers to be able to carry out each task, some context information may be required. For example, the student, from the first scenario, may want to know where the buildings are located in the university campus. Also, the office hour information showing opening and closing times of each service centre will help him find the optimum route. The same principle applies to the asset management officer from the second scenario. Extra information such as how far rooms are located from each other, how many assets are to be collected at each location, etc. may help him schedule the field trips efficiently.

The resources to be considered by the workers may differ depending on the nature of the tasks, the skill level of the workers, or the kind of roles the workers play in an organisation. In deed, we believe that a thorough study into the requirements of the workers in making decisions as well as a survey of effective visualisation techniques have not been explored. This is an important part of our current on-going investigations, in which we look at identifying various types of resources that a worklist can provide to help the workers carry out the tasks. Also,

² By rejection of tasks, we mean choosing not to accept the task. Such task can be delegated, suspended, or re-allocated by the workflow system.

³ The readers are noted that these examples are simplified for illustration purposes.

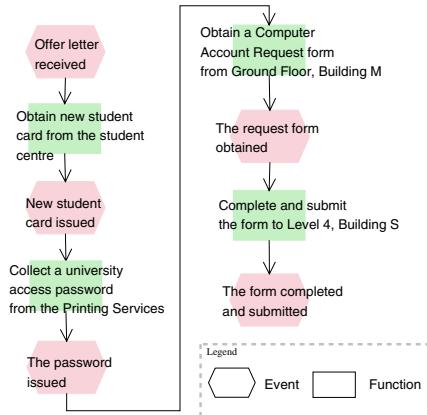


Fig. 1. First Case: Event-Process-Chain diagram of the student process

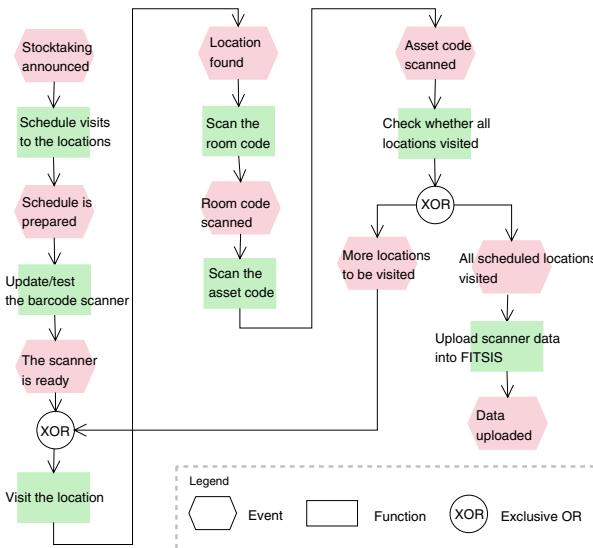


Fig. 2. Second Case: Event-Process-Chain diagram of the stocktaking process

we examine multiple aspects of the issue involving the nature of tasks, workers background, etc. Ultimately, the study aims to develop a detailed analysis of resources that will help identify the factors and their inter relationships that affect the worker's worklist management.

In this paper, we identify a few common types of resources that may have some generic applications. A list of the resources we have identified so far is presented Table 1⁴.

⁴ More resources will be added as the analysis becomes more complete.

Table 1. Generic types of resources

Resources	Descriptions
space	size or spatial information relevant to the tasks. It may be a diagram showing available storage rooms and their sizes, or meeting rooms and their capacities. This type of resource may be used to determine, for example, where 20 computers should be stored. It is separate from location, as sometimes the visualisation may not relate space with actual location of the space – space to store computers, but not interested in where.
materials	materials or consumable information relevant to the tasks. It may be an inventory list of materials to be used in the task, and whether you have enough of those things: number, volumes, weights, dimensions.
equipment	equipment information relevant to the tasks. It may be, for example, an inventory of barcode scanners showing their availabilities.
services	internal or external services information. It may be a list of travel booking agencies, printing services, or messaging services and their contacts/availabilities.
time	any “time” information relevant to tasks. It could be deadlines (ie., the time each task should be completed by), opening hours (ie., the time a particular service, for example a printing centre, is available) or a simply calendar showing working days. This type of resource will be useful in one Rs planning of the sequence of task executions.
location	geographical “location” information relevant to tasks. It could be a map of a campus showing locations of university facilities, a floor plan of an office block, or a diagram showing relative distances between locations. This type of resource also can be used in scheduling of tasks. We separate this resource from space as our model uses location in both the sense of a resource (maps), and as a generic place holder for the work item location in the visualisation (grid layout).
people	information about people and their roles in an organization. It could be an organizational chart showing roles and responsibilities of people. This type of resource may be used in finding the right person to seek for specific help or delegate a task to.
active worklist	current (active) tasks that are being carried out by the worker. This type of resource will help the worker determine the desirable workload, and effectively manage the current/future tasks. This can be represented by an arbitrary grid arrangement, where each cell represents a task to be performed, and may or may not contain other resource information regarding the task.

4 Mapping Resources to a Worklist Visualization

In this section, we describe our generic framework built for worklist visualization based on the resources we presented earlier. Again, for illustration purpose, we choose the four resources we described in the previous section; time, location, people and active worklist.

The visualisation framework is based on a layered approach, in which a background and overlay planes are used. A 2D representation of any of the resources forms the background layer. For example:

- The time resource uses a time line form of representation (eg., GANTT chart);
- The location resource uses a map representation that shows whereabouts and distance between locations (eg., Street maps)⁵;
- The people resource uses a chart or diagram form of representation (eg., organisational charts);
- The active worklist resource uses a regular grid spatial arrangement; an arrangement different to irregular different layouts like maps.

The overlay plane consists of the tasks in a worklist. Each task is given (x,y) coordinates in relation to the background, which indicates the resource information allocated to the task.

Let us consider the first scenario. The student has four tasks to complete and he has to visit different service centres in the campus;

1. Obtain new student card from the student centre
2. Collect university access password from the printing service
3. Obtain a computer account request form from Ground floor, Building M
4. Complete and submit the form to Level 4, Building S

With our worklist visualization framework, we can present the four tasks as shown in Fig. 3. On the left-hand side of the figure, the background layer shows a campus map (ie., a form of the location resource). Each task (shown as a (round) coloured icon in the figure) is given (x,y) coordinates in relation to the map which indicates the location where the task is supposed to be carried out. For instance, the first task (a green icon on Building A) is placed on top of the building where the student centre is situated, and so on. The same worklist can be presented from the point of view of other resources. On the right hand side of Fig. 3, the chart diagram illustrates a visualisation of the same worklist from the perspective of the people resource. In this view, each task is given (x,y) coordinates in relation to a chart of organisational units. It shows which organisational unit is responsible for administrating each task. For instance, the first task (a green icon on Student Centre) is placed on top of the administration unit the student needs to contact if he/she needs help.

All of the worklist items on a case appears on every “view”. If the system selectively placed worklist items on a view, then the worker may miss items, or get confused by the changes from view to view. In addition, the item may require assessment from a number of resource viewpoints before being accepted or rejected by the worker. Therefore, we have the entire worklist shown on each view. The workers can easily switch from one view to another.

The framework is simple yet generic in that any types of resources can be presented using the overlaying technique. The same worklist can be viewed from different resource perspectives.

⁵ Note that this could be different from the spatial map used to represent the space resource that shows occupying space, eg., building plan.

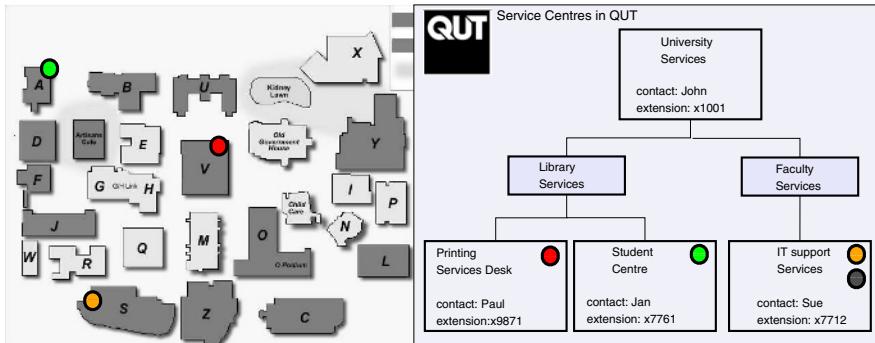


Fig. 3. Illustration of background and overlay structure of visualisation

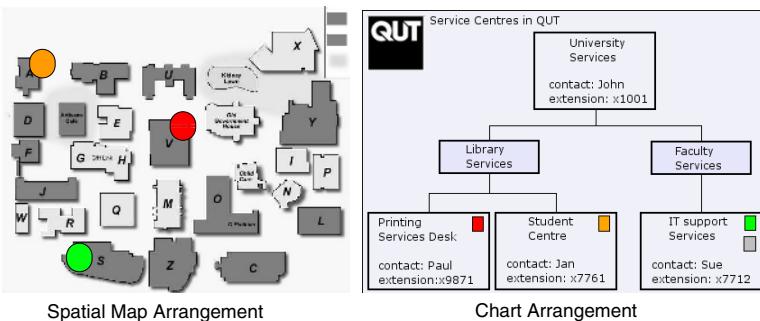
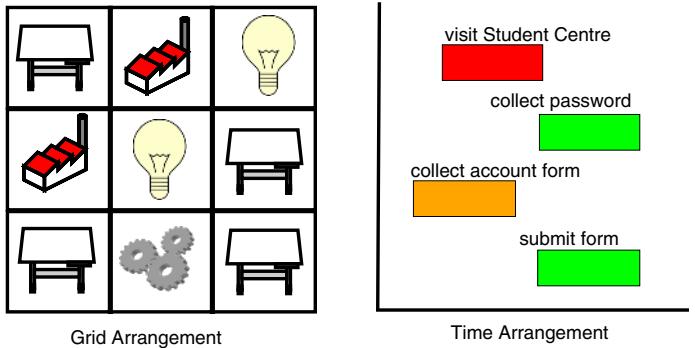


Fig. 4. Examples of the visualization categories

While having separate “views” can be useful, there is a requirement for cross-over with regards to the background-foreground information. For instance, the time resource may be shown as an overlay plane resource on a grid background; via the use of numbers overlaid on the grid locations. In Fig 4, examples of dif-

Table 2. Table enumerating the broad categories of resources, their use in worklist choice decisions and appropriately mapped visualisations

Resources	Worklist Choice Function	Visualisation
time	To compare the relative start and finish times for each task and insert it into the worklist at appropriate moments if time resources are available, either by leaving the task as whole, or dividing it into smaller components for insertion into small time gaps.	Gantt Chart showing all available tasks on a time line in stacked manner to identify insertion points for the worklist components.
location	To compare the spatial locations of tasks to be performed for logistical purposes.	Map detailing the arrangements of tasks in space, to aid the worker in identifying efficient ordering of the work.
people	To show visualisations of number of people available for task and their capabilities to assess who is appropriate for the task.	Overlays of people available to meet task with encoding of match between people and the tasks – colours/textures, including hierarchical views, social network views.
space	To compare the space resources required for a task to the space resources available.	Map detailed with space allocations showing empty spaces at certain times.
active worklist	To see a list of active tasks which can be checked out; user chooses according to the number of tasks they are able to perform.	Worklist dialogue, with overlaid data for comparison and choice of task.
materials	To view materials to be used in the task, and whether you have enough of those things: number, volumes, weights, dimensions	Overlays of information onto base background for any of the visualisations to compare materials with other materials available at that location, or a calculated indication of ability to meet this role.
equipment	To view equipment to be used in the task	Overlays of information onto base background for any of the visualisations to compare Equipment count with equipment available at that location, or a calculated indication of ability to meet this role.
services	To view availability of services from internal or external agencies in order to complete the task	Overlays of information onto base background of the availability of these services to meet the task.

ferent views are shown; a worklist (in a grid format), location map and timeline. In each case the task is given a coordinate to arrange it in 2D on the surface of the background.

Each is a representation that can be used within a push-based task dissemination system to decide about task choices, with regards to the relevant resources. They can be turned on and off by the designer of the workflow visualisation to allow or deny access to extra information regarding tasks. Each one can be modified to suit a particular application area, thus leaving room for development of novel visualisations tailor-made for different applications [1]. So the general rules are able to be modified but can still be encapsulated in the development of a set of visualisation tools. Table 2 maps each resource type to an appropriate visualisation. The table is by no means exhaustive, and is only limited by the number of application areas intended for the visualisations. We present some that are appropriate for general visualisation applications [2].

5 Worklist Management with Visualisation

In this section, we explain how the workers can interact with the visualized worklist. First, we introduce a generalized algorithm that workers use to manage their tasks, and then we explain the interactions between the workers and the visualised worklist.

5.1 A Generic Algorithm for Managing a Worklist

The resource view specified by [16] treats a resource as being human or non-human, and will have tasks directed to them by the workflow system. In this paper, for the sake of clarity, we use the term worker to differentiate human resources from non-human resources. Worklist items are distributed to workers within an organisation according to the process illustrated in the life cycle diagram in Fig. 5.

Inherent in this distribution process by the workflow system is the choice by the system of whom to give the task, via the offer actions. The workflow system

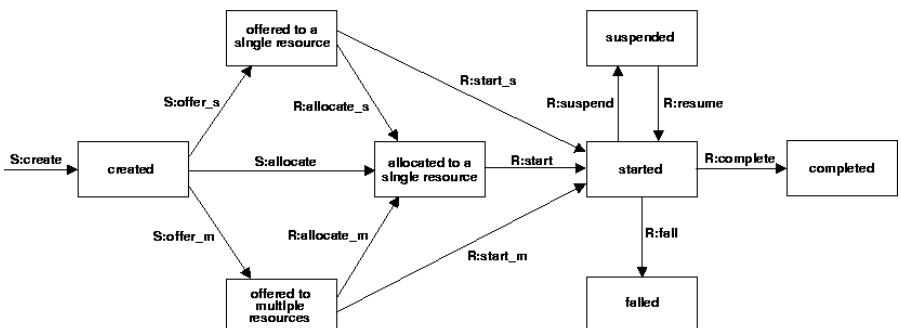


Fig. 5. Illustration of task life cycle; modified from [16]. Each box is the state of the task in a running workflow case. The prefix S and R refer to actions enacted by the Workflow System and the Resource (Worker) respectively.

will have a resource view that evaluates the capabilities of the intended recipient of the task. Furthermore, some of the resources will have an optimisation performed upon them by the resource view; eg. time and space, and will have this information offered to the worker to help them with their decision. The worker upon receiving the task, must make a decision for themselves about accepting or not accepting the task. This process is out of the control of the workflow system, as it only can *push* tasks to the worker to request acceptance. The workers' responses have been characterised by *detour process*. A worker may *delegate* – hand to another worker, *de-allocate* – reject a task, *re-allocate* – task is handed to another worker by the system, *suspend/resume* – halt and then recommence a task allocated to a worker.

The question for the worker is the choice of adding or rejecting (ie., detouring) a task from his/her worklist. The task allocation can be a *push* or *pull* approach; push being system selected, pull being worker selected. Assuming a more pull oriented model of worklist task selection, our resource centric views of a worklist will aid the worker in this worklist management task, as they are able to decide which item to choose based upon critical resource issues.

The workflow system may offer a number or only one instance of the task to the worker, and at this point the worker may decide to perform the task by checking them out and adding them to a list of active tasks, or the user may decide to return the task to the unallocated pool via the detour process. Furthermore, the worker upon completion of the task checks the task in, thus removing it from the active checked out worklist. This task acceptance process may be represented by the following formula for the acceptance process, them being the check out processes respectively:

$$W_r = W_r \cup \{I\} \iff C_{W_r, T} > C_{\{I\}, T} \quad (1)$$

where:

- W_r is the set of worklist items for worker(s) r ;
- I is the new worklist item to be added;
- $C_{x,y}$ is the capability for the task(s) x of type y ;
- T is the type of resource being processed (eg. Computer Equipment).

So, at any stage a worker will make a decision about whether to add a worklist item to its set of worklist items, by looking at the capabilities of the worker for the present worklist as compared to the requirements of the new task. This can be automated, but the worker must be allowed to make such decisions as well, in order to promote a healthy attitude within the workforce. But it must be recognised that people will simply decide not to do a task, if they do not want to or decide to prioritise using undefined criteria. Furthermore, these visualisations may give information to the worker regarding the reasoning behind the choice of been allocated the task, and so the worker is left in an informed state about the reasons for work allocation.



Fig. 6. Illustration of an aggregated icon made up of single task icons. The example shows a task titled “Prepare Stock Check Report” with zero checked in, one checked out, three available and one task unavailable.

5.2 Interacting with the Visualized Worklist

In the visualised worklist, each task is represented by a coloured icon. Some workflow systems support the generation of a number of instances of tasks, that may be disseminated to workers [17]. Thus at times an aggregated icon has to be used to represent multiple instances of the task in question. Fig. 6 shows an example of a task with multiple instances. An aggregated icon is shown with four icons with numeric information regarding the number of instances and their status within the system. The state of any delivered task at one time may be the following: inactive, available, checked out and suspended, and included is the colour we have mapped to the state using the traffic light metaphor of red, green and amber:

- Inactive – unavailable to the worker (grey);
- Available – available to the worker to check out (amber);
- Checked Out – has been checked out by the worker (green);
- Checked In – has been checked in and completed by the worker (red);
- Suspended – has been checked out by the worker, is still incomplete, but checked in to the user (amber – dashed);

We use the traffic light metaphor due to its intuitive mapping to the status of the tasks: green active (go), red completed (stop) and amber available (in between go and stop). Furthermore, the available state is refined to have a dashed amber appearance for those items that are suspended, and so the dashed appearance represents a partially completed task.

The worker interacts with the icons in a similar manner to previous worklists, by clicking on the icons to check out available tasks, and by clicking on checked out icons to check in completed tasks. Whenever appropriate, a form will be presented by the workflow system, to obtain data from the worker.

6 Implementation

A major test of any workflow visualisation approach is its ability to be incorporated into a modern client server-based workflow system. We have built a prototype

of the proposed visualisation framework, and interfaced it with the workflow system YAWL. This section discusses the system architecture and implementation in detail.

6.1 The YAWL Environment

Our implementation is based on the open source workflow environment named YAWL (Yet Another Workflow Language), which is a research initiative at Queensland University of Technology [4,17]. YAWL is based on a set of workflow patterns developed via analysis and comparison of a number of commercial workflow systems. It provides powerful and formal workflow description language, as well as an execution environment.

To understand the architecture of our visualisation framework, let us first present the overall architecture of YAWL. Workflow specifications are created in the YAWL designer which is a graphical editor, and deployed to the YAWL engine. The engine performs verification of the specifications and stored them in the YAWL repository. The specification can be loaded and launched for execution via the YAWL manager, and is hereafter referred to as a schema. The execution itself is managed by the YAWL engine.

The YAWL engine interacts with the components labelled as *YAWL services* through *Interface B*. The YAWL services (worklist handler, web services broker, interoperability broker and custom YAWL services) are based on the web services paradigm and all are abstracted as services in YAWL.

How the engine communicates with the YAWL worklist handler is of particular interest in our work. The worklist handler is the component that is responsible for dispatching tasks to the workers. Through the worklist handler, the workers accept tasks and mark their completions.

In conventional workflow systems, the worklist handler is part of the workflow engine. However, in the YAWL environment, it is a separate component that interacts with the engine through Interface B. Through the interface, a custom service or application can be developed to extract worklist information for display in whatever manner is required.

6.2 Worklist Visualisation Architecture

Based on the existing YAWL architecture, we have developed a new type of YAWL worklist handler which interacts with the engine through Interface B. The overview architecture is shown in Fig. 7. It has capabilities to (i) display the visualised resources and (ii) dispatch tasks like a normal worklist handler. The architecture consists of two components which have designed and partially implemented: a visual worklist handler and a visualisation designer.

The visual worklist handler can view multiple cases of running workflows, with multiple resource-centric views matched to the requirements devised by the YAWL schema designer. The worker loads the cases and is presented with a list of tasks, and a tabbed view list to switch between difference representations of

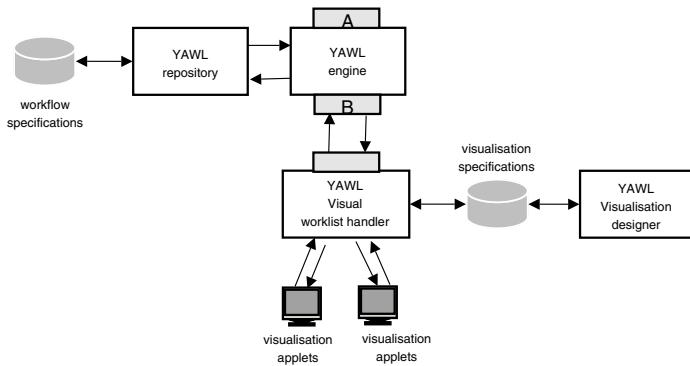


Fig. 7. YAWL Visualisation Framework: Overall architecture

the worklists. In the following two sections we describe the two components, and illustrate them with mock ups containing partially developed examples.

6.3 YAWL Visualisation Designer

The designer application is the most complete at this stage. It is designed around the structure of the visualisation approach we have developed, and is implemented in Java, as is the rest of the YAWL implementation. The visualisation designer allows the user to load Scalable Vector Graphics (SVG) files as backgrounds and icons for the overlay planes. This allows easy modification of images via other drawing tools. The SVG component of the designer is managed by the BATIK Java package [18]. This is thus an implementation of the task coordinates scheme we detailed earlier. This designer allows the easy outlaying of tasks as icons across the background in the program.

The process of designing a visualisation view for a schema is as follows:

1. First decide on the background and overlay images, editing them in a separate tool and saving them as an SVG file;
2. Decide on the spatial arrangement of the tasks to be displayed according to the resources that need to be analysed, for example a map for logistics on QUT campus that will help a worker to decide where to perform their tasks;
3. Load the workflow schema into the editor to obtain the tasks in the system, which appear in a mouse menu on a right click at the chosen location on the background;
4. Load the background image;
5. Set the current icon to be used by choosing from the list in a dialog;
6. Move pointer to location of worklist item and right click to choose a task, and icon, repeating for all worklist items.

Fig. 8 illustrates the major components of the visualisation designer user interface via the stocktaking example on campus map (Fig. 2). The large window (right) is the main window for visualisation design, and the smaller window (left)

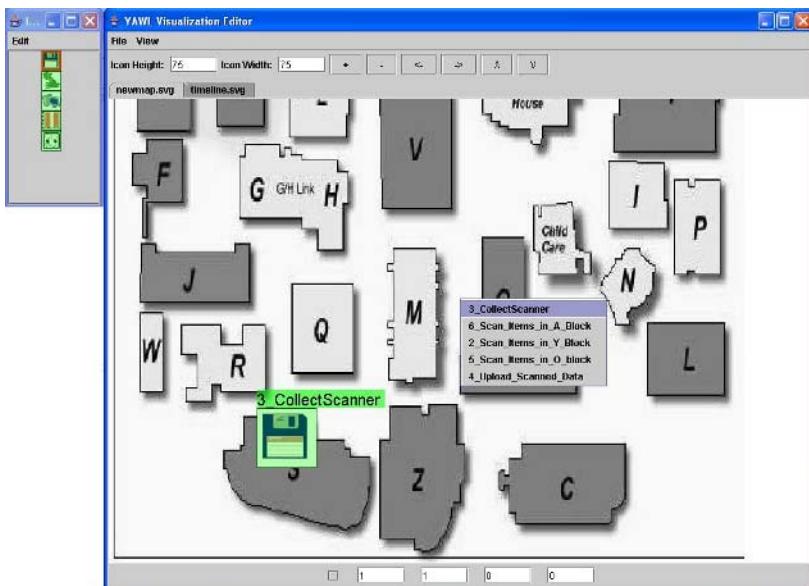


Fig. 8. Yawl Visualisation Designer: main components

shows a list of potential icons to be placed at locations on the visualisation. Each view is placed into a tabbed list, as they are to be displayed in the visualisation agent. The menu is displayed using a mouse right click, showing the tasks defined in the schema. The icon can be placed at the location of the right click of the mouse, or using actual coordinates in the text entry boxes at the bottom of the screen. The icon at the bottom left of the image is the current task icon, “CollectScanner” and is shown using a disk icon.

This visualisation design information is stored in an XML file that defines an arbitrary number of views per schema, and the task icons, gained from the number of tasks within the YAWL schema. This file is then read by the Visual Worklist Handler to form the visualisation structure for communication to the YAWL engine. The following is a snippet from a visualisation specification. A specification may have a number of <view>s, and each view may have a number of <task>s. A view is associated with a background representing a resource. Each task is assigned a color for the description, coordinates, and an icon.

```
<specification id = "TSSstockTake.ywl"
    uri = "file:/D:/Yawlstuff/batik/demo/TssStockTake.xml">
<view id = "file:/D:/Yawlstuff/batik/demo/map-1/newmap.svg">
<task id = "3_CollectScanner">
<color> -16777216</color>
<coordX> 240</coordX> <coordY> 760</coordY>
<icon width="75" height="75">file:/D:/Yawlstuff/demo/floppy.svg</icon>
</task>
</view>
</specification>
```

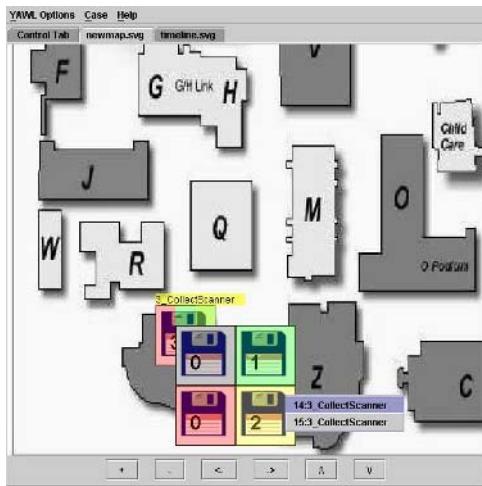


Fig. 9. Screen dump of a running visualisation handler, showing a campus map visualisation with an icon showing the “CollectScanner” icon from the PC Stocktake example as an available worklist item in orange amongst multiple instances

We have implemented the beginnings of a visualisation editor and visualisation viewer, which we show in this paper. In a final implementation, additional resource information will be selected from the resource view of the YAWL schema as it is running. For now we are able to design worklists arranged according to grid, spatial and time arrangements.

6.4 YAWL Visual Worklist Handler

Worklists are disseminated in YAWL via the default worklist handler as simple dialogs containing lists of tasks, with no other resource information being displayed. We have begun implementing a visual worklist handler that is an extension of the default handler. The YAWL workflow implementation is structured around a component architecture that communicates via XML formatted commands. Thus the worklist handler is able to utilise the B interface to the running YAWL case in the same manner as the default worklist handler. The visual worklist handler is able to execute the visualisation developed with the designer that is stored in a file (see Fig. 7).

The new worklist handler allows a more intuitive mapping of task coordinates to the check in and check out process. The user is able to check items in and out by simply clicking on the potential worklist item in its location on a map or hierarchy diagram. right image is a mock up of the student enrolment example visualisation running within the visualisation handler.

With a spatial organisation to the tasks, the person doing this registration process can evaluate the task, using the map to make a decision about the acceptance of the worklist item in consideration of the location resources.

7 Conclusion

We have described the beginnings of a thorough analysis of workflow visualisation; its theoretical basis, resource centric approach and appropriate visualisation techniques. Analysis in our paper showed to use these techniques within a typical workflow system. The task coordinate approach was described, showing how this can be generalised across a number of visualisations using a background and overlay approach. We have also begun the development of a visualisation development environment, with an editor and visualisation agent that uses SVG files and is easily integrated into the YAWL workflow system created by the BPM group at QUT. Thus we have indicated that this visualisation approach can be used within a fully featured workflow environment.

Further analysis will continue to refine the visualisation mappings to produce a knowledge base for development of visualisations within workflow applications. In particular, there will be refinement of the broad categories of resources into more fined grained categories to derive a rule-base for an intelligent design agent to be incorporated into the visualisation designer. Evaluation experiments will be performed within a case study in order to ascertain the effectiveness of the resource centric visualisation approach with users of workflow tools.

In addition, we will exploit the latest resource view developments that are being implemented within YAWL, to enable the run time specification of resources associated with a task, and thus extend the implementation to include automated run time visualisations of resources such as people and equipment, associated with the tasks. We will thus extend the visualisation editor and agent to accommodate these resources in a structured manner, according to our table of visualisation mappings.

Acknowledgement

This project is partially supported by a QUT Faculty of Information Technology collaborative grant. We acknowledge the programming assistance provided by Tore Fjellheim and Guy Redding, who programmed the visualisation editor and agent applets and integrated them into the YAWL workflow system. Their dedication and hard work towards implementing this project have been greatly appreciated. We thank Prof. Wil van der Aalst for his invaluable contribution.

References

1. Keller, P., Keller, M.: *Visual Cues*. IEEE Press., Piscataway USA (1993)
2. Tufte, E.: *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, USA (1983)
3. Andersson, T., Bider, I.: Introduction of bps systems into operational practice: Achievements and setbacks. In: Proc. of BPMDS, Riga, Latvia (2004)
4. Aalst, W.M.P.v.d., Aldred, L., Dumas, M., Hofstede, A.H.M.t.: Design and implementation of the YAWL system. In: Proc. of CAiSE, Riga, Latvia (2004)

5. Luttighuis, P., Lankhorst, M., van de Wetering, R., Bal, R., van den Berg, H.: Visualising business processes. *Computer Languages* (2001) pp.39–59
6. UNISYS: 3d visible enterprise (2004) www.3dvisibleenterprise.com/3dv/.
7. Schonhage, B., van Ballegooij, A., Elliens, A.: 3d gadgets for business process visualization:a case study. In: *Symposium on Virtual Reality Modeling Language*, Monterey, California, ACM Press (2000) pp.131–138
8. Systems, I.S.: Interactive software (2004) www.interactive-software.de/.
9. Grohn, M., Jalkanen, J., Haho, P., Nieminen, M., Smeds, R.: Visualizing human communication in business process simulations. In: *Visual Data Exploration and Analysis VI*, San Jose, SPIE-Int. (1999) pp.140–148
10. Hsu, C., Yee, L.: Model-based visualization for enterprise information management. In: Proc. of 4th Annual Conf. on Artificial Intelligence, Simulation, and Planning in High Autonomy Systems, Tucson, Arizona (1993) pp.324–327
11. Gershon, N., Eick, S.G.: Information visualization. *Computer Graphics and Applications* **17** (1997) pp.29–31
12. Wright, W.: Business visualization adds value. *Computer Graphics and Applications* (1998) p.39
13. Latva-Koivisto, A.: User interface design for business process modelling and visualisation. Technical report, Department of Computer Science, Helsinki University of Technology, Helsinki (2001) Masters Thesis.
14. Leroux, H., Exton, C.: Coope: a tool for representing concurrent object-oriented program execution through visualisation. In: Proc. of 9th Euromicro Workshop Parallel and Distributed Processing. (2001) pp.71–76
15. Jennings, N., Norman, T., Faratin, P.: Adept: An agent-based approach to business process management. *ACM SIGMOD Record* **27** (1998) pp.32–29
16. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow resource patterns: Identification, representation and tool support. In: Proc. of CAiSE, Porto, Portugal, Springer Verlag (2005) (to appear).
17. Aalst, W.M.P.v.d., Hofstede, A.H.M.t.: YAWL: Yet another workflow language. *Information Systems* **30** (2005) pp.245–275
18. Batik: Batik svg toolkit (2005) <http://xml.apache.org/batik>.

CoopFlow: A Framework for Inter-organizational Workflow Cooperation

Issam Chebbi and Samir Tata

Department of Computer Science, GET/INT,
(Institut National des Télécommunications),
9, Rue Charles Fourier 91011 Evry, France
{Issam.Chebbi, Samir.Tata}@int-evry.fr
<http://www-inf.int-evry.fr/~tata>

Abstract. The work we present here is in line with a novel approach for inter-organizational workflow cooperation spanning several organizations without being managed by one physical organization. Our approach consists of three steps: workflow advertisement, workflow interconnection, and workflow cooperation. Hence, to implement a virtual organization it is important to provide a mechanism whereby organizations can advertise their workflow parts, other organizations can look at them and cooperate these with their own workflows. In this paper, we present *CoopFlow*, a workflow cooperation framework, supporting dynamic plugging and cooperation between heterogeneous workflow management systems (WfMS). Can be connected to *CoopFlow* any WfMS that is able to invoke external applications (programs, Web services, etc.) and that allows external applications to invoke any step within a workflow it manages. *CoopFlow* presents many advantages. First, it provides powerful ways for inter-organizational workflow cooperation and code reduction. In fact, partners can change their WfMS without changing the global proxy behaviour. Furthermore, it permits a dynamic interconnection and disconnection of participating organizations. In addition, it preserves the privacy and autonomy of process participants by reducing inter-visibility as tiny as the cooperation needs based on the view principle. Finally, our framework preserves established workflows : participants don't modify their internal systems. Instead, they have just to implement a proxy to integrate *CoopFlow*.

1 Introduction

In context of globalization, a high competitive pressure characterizes the general situation on businesses. Competition can lead to intensive re-structuring of organizational structures and processes to make production and services more efficient and less expensive. Additionally, new forms of inter-organizational collaboration between organizations may emerge. In this case, organizations especially Small and Medium sized Enterprises (SMEs), cooperate to fulfil conditions of complex projects. Thus, partners with complementary competencies

and knowledge can be gathered to carry out projects, which are not within the range of only one organization (especially SMEs): cooperation allows each partner to benefit from knowledge of the other partners in the virtual organization.

Parallel to this evolution, organizations are increasingly utilizing process-aware information systems to perform their business processes in an automated¹ way. Based on such information systems, organizations focus on their core competencies and access other competencies through cooperation, moving towards a new form of network known as virtual organization.

There is still no agreed-upon definition of virtual organizations. As for us, we define a virtual organization as a set of partners (“real organizations”) distributed in time and in space sharing resources and competencies and cooperating to reach some shared objectives using information technologies [2]. In this context intention, partner workflows are not carried out in an isolated manner, but interact during their execution, while sharing common data in a coordinated way. Coordination brings a synergy that contributes to the improvement of each partner work performances.

In this paper, we focus our work especially on workflow cooperation in a context of virtual organizations. Cooperation in this case is temporal and structured by a common objective and for the duration of a project. We think that the number of virtual organizations is huge and that they are an important market for Internet technology which allows short duration and low cost connections between partners ((especially SMEs) who cannot, in general, establish specialized and “hard coded” cooperations. In line with an approach we have proposed for workflow cooperation [3, 4, 5], we propose in this paper *CoopFlow*, a workflow cooperation framework, that in addition to interoperability supports workflow interconnection, cooperation, monitoring and control in flexible and efficient manner.

Our work aims at supporting cooperation in inter-organizational workflows. one of our fundamental objectives is to address tow main issues. (i) How to provide a mechanism whereby organizations can advertise their workflow parts, other organizations can look at them and, finally, cooperate these with their own business process? (ii) How to execute an inter-organizational workflow spawning several organizations without being managed by one physical organization?

The remaining of this paper is structured as follows. In Section 2 we give the requirements for inter-organizational workflows as well as related work in the area of inter-organizational workflows. In Section 3, we summarize our bottom-up approach to inter-organizational workflows cooperation. In Section 4, we describe the framework architecture for cooperation enforcement and detail its components. Section 5 is devoted to the framework implementation solution. Conclusions and perspectives are presented in Section 6.

¹ A workflow is seen as an “automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules” [1].

2 Requirements and Related Work

In this section, we present requirements we have to deal with in order to support workflow cooperation in virtual organizations as well as related work.

2.1 Requirements for Inter-organizational Workflows Cooperation

In a context of workflow cooperation one has to deal with three main requirements: flexibility support, privacy respect principle, and established workflow and WfMS preservation.

Flexibility support: Since there is some dynamic structure to the cooperation between the organizations (constant evolution of the partner's set, dynamic join/leave of participating organizations), we think that cooperation description is difficult (if not impossible). Therefore, solutions for workflow cooperation have provided whatever means possible for dealing with flexibility.

Privacy respect principle: On one hand, cooperation needs a certain degree of workflow inter-visibility in order to perform interactions and data exchange. On the other hand, cooperation may be employed as a cover for organizations to internalize the know-how of their partners. The question here is how to best preserve the know-how of each partner and capitalize on the accumulated experience and knowledge to allow cooperation and to improve productivity. In order to preserve privacy and autonomy of process participants, one must reduce workflow inter-visibility to be as little as the cooperation needs.

Established workflow and WfMS preservation: For enabling cooperative organizations to integrate their disparate workflows it is necessary to allow them to use established workflows. When planning projects, it's important to note that any changes to established workflows or systems that manage them will cost money and time. Therefore, if organizations are to achieve the efficiencies and reduction in costs that the cooperation promises especially if they are SMEs, approaches for workflow cooperation must fully integrate pre-established workflows.

2.2 Related Work

Recently research on workflow management has focused on inter-organizational cooperation issues which have been addressed by numbers of approaches using the notion of contracts, workflow interoperability or specification languages.

Workflow Contracting. The inter-organizational cooperation problem has been addressed by using the notion of contracts to define the business relationships between organizations. Examples of this type of cooperation support are *COSMOS*[6], *TOWEC*[7], *CrossFlow* [8], *WISE* [9, 10] and *ebXML* [11, 12]. In the following three paragraphs we briefly discuss *CrossFlow*, *WISE*, and *ebXML*.

CrossFlow: The CrossFlow [8] project investigates some issues which are concerned with business processes crossing organizational boundaries. A contract-based approach is used to define the business relationships between the organizations. Within this contractual basis, inter-organizational processes can be

defined and performed. However, the approach does not support arbitrary public processes and no standard definition language and semantics is provided for the enforcement of contracts between two enterprises. In addition, all enterprises involved are required to use the same software for contract enforcement.

WISE: The *WISE* [9, 10] (Workflow based Internet SErvices) project aims at designing, building, and testing commercial infrastructures for developing distributed applications over the Internet. It proposed a framework to compose a virtual business process through process interfaces of several enterprises. This architecture provides means to define, enact, and monitor virtual enterprises business processes, as well as to manage context aware communication among process participants. It includes an Internet workflow engine to control the business process execution, a process modeling tool to define and monitor processes, and a catalogue tool to find the building blocks for the processes. The accessibility over the Internet makes *WISE* scalable and open but service descriptions and the service catalogue are not in line with standards. Moreover, the centralized workflow engine inhibits dynamic selection and exchange of partners since all participants have to comply with stipulated interfaces.

ebXML: *ebXML* [11, 12] aims at providing a framework for business to business transactions. *ebXML* can be seen as a global electronic market place where enterprises of any size, anywhere can find each other electronically and conduct business through exchange of XML based business messages. It supports re-usable data types, interorganizational transactions, and profiles and agreements. It offers interaction primitives to support timing, security, and atomicity properties. Capabilities of an individual party is described in term of Collaboration Protocol Profile (CPP) which is stored in *ebXML* registry (*i.e.*, Business partners can find each other's CPP through registry). Capabilities that trading partners have agreed to use to perform a particular business collaboration are described in term of Collaboration Protocol Agreement (CPA).

In the contracting approaches, it is clear that support for inter-organizational workflows can be improved substantially. In fact, flexibility and workflow preservation requirements are not met. In the CrossFlow approach, for example, all enterprises involved are required to use the same software for contract enforcement. Moreover, the approach does not support arbitrary public processes.

Workflow Interoperability. In [13], the author presents some forms of workflow-interoperability and focuses on capacity sharing, chained execution, subcontracting, case transfer, loosely coupled, and public-to-private architectures. In the following we present chained execution and public-to-private approaches.

Chained execution: In the chained execution approach, the process is divided into subsequent phases and each business partner takes care of one phase. The workflow process is split into a number of disjunctive sub processes executed by different business partners in a sequential order [14]. This form of interoperability is only useful for applications where the process is composed of sequentially

ordered parts. Nevertheless, it was generalized into an approach to distributed workflow execution where parts are inter-mixed [15]. However, this last approach is static since it starts from a global centralized workflow where all activities are known a priori and assumes that for each activity there exists an assignment to a department or business unit of the enterprise.

Public-to-private: In the public-to-private approach, a common public workflow is specified and partitioned according to the organizations involved by private refinement of the parts based on a notion of inheritance. Each partner has a copy of the workflow process description. The public-to-private approach consists of three steps. Firstly, the organizations involved agree on a common public workflow, which serves as a contract between these organizations. Secondly, each task of the public workflow is mapped onto one of the domains (*i.e.*, organization). Each domain is responsible for a part of the public workflow, referred to as its public part. Thirdly, each domain can now make use of its autonomy to create a private workflow. To satisfy the correctness of the overall inter-organizational workflow, however, each domain may only choose a private workflow which is a subclass of its public part [16].

Problems to be encountered on the way to workflow interoperability include mainly autonomy of local workflow processing, confidentiality that prevents complete view of local workflow [17], and especially flexibility that needs no definition of a global workflow that defines cooperation between local workflows. In addition, a drawback of the approach presented in [16], is the lack of the preservation of preestablished workflows. In fact, in this approach, one has to look for which rules, in what order and how many times one has to apply in order to match the preestablished workflow with the public part which is deduced from partitioning of the public workflow. If not impossible, this is hard to do. Moreover, there is no defined procedure to do that.

Specification Languages. To specify workflows interoperability, big efforts have been made and many specifications have been proposed. We cite among others *SWAP*, *Wf-XML* [18, 19] and *ASAP* [20]. In the following, we present a very brief survey of the *Wf-XML* and *ASAP* specifications.

Wf-XML: *Wf-XML* [19] is a simple process request/response protocol that defines how to install a process definition into a process engine for execution. Although it enhances some of its predecessor's capabilities by providing a structured and well-formed XML body protocol and synchronous or asynchronous message handling and a transport independent mechanism, *Wf-XML* is insufficient for workflows cooperation and suffers from a number of lacks.

Asynchronous Service Access Protocol (ASAP): *ASAP* [20] is an OASIS standard that enables Web services to be used in business processes where a response to a service request might take a long time, usually because there is human intervention required. When used in conjunction with the new version of *Wf-XML*, *ASAP* allows different companies to link up their business process management systems, regardless of the BPM vendor and without additional programming.

Despite their diversity, it is clear that the workflow interoperability specifications don't offer enough means for cooperation and lacks functionalities to control and enforce cooperation policies in inter-organizational workflows.

3 An Approach for Workflow Cooperation

In line with our fundamental objective to support a virtual organization we have developed a novel approach that consists in three steps: (1) advertisement of parts of organizations' workflows that could be exploited by other organizations, (2) interconnection of cooperative workflows and (3) workflow cooperation and monitoring according to cooperation policies (rules). In this section we present this approach using an example. Next sections focus on the *CoopFlow* architecture and implementation.

3.1 Example

For illustration, consider an example involving two business partners: a client and a product provider. Figure 1-(a) presents the workflow of the client using Petri nets as specification language [21]. First, the client sends an order for a product. Then she receives a notification. When the product is ready, the client receives the delivery and the invoice. Finally, she pays for the ordered product. Figure 1-(c) presents the workflow of the product provider. First, the provider waits for an order request. Then he notifies the client that her order was taken into account and he assembles the components of the product. After that, two cases can happen: the client is a subscriber (she often orders products) or she is not. In the first case, the provider sends the product and the invoice and waits for the payment. In the second case, the provider sends the invoice, waits for the payment and then sends the product. Filled activities, in Figure 1, are the ones that cooperate with external partners. The examples given here only show cooperation between two partners. Nevertheless, the contribution we propose here also addresses cooperation between more than two partners.

The approach we present here is inspired by the Service-oriented Architecture that requires three operations: publish, find, and bind. Service providers publish services to a service broker. Service requesters find required services using a service broker and bind to them. Accordingly, our approach consists of three steps: workflow advertisement, workflow interconnection, and workflow cooperation.

Figure 2 presents the sequence diagram of the different steps of workflows cooperation. The diagram involves a cooperating partners set, a registry, participants, and a contracting authority ensuring the cooperation monitoring and control. We identify three logical blocks that can be implemented by one or several physical entities. These blocks correspond to workflow advertisement, workflow interconnection, and workflow cooperation.

3.2 Steps for Workflow Cooperation

Step 1: Workflow Advertisement. For building an inter-organizational workflow, each partner has to advertise, using a common registry, its offered and

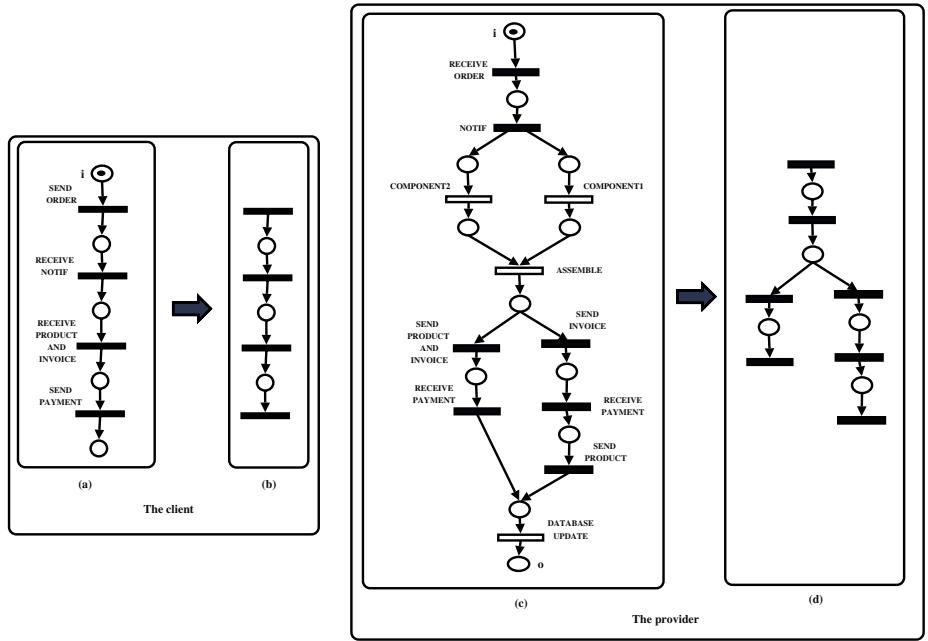


Fig. 1. Workflows and cooperative interfaces of the client and the provider

required activities within their workflows. In the first block of the Figure 2, partners publish their cooperative activities, control flows and data flows, that we call cooperative interface, into the registry.

Figure 1-(b) presents the workflow cooperative interface of the client and Figure 1-(d) presents the workflow cooperative interface of the provider. Broadly speaking, a workflow cooperative interface is a projection of a workflow on the cooperative activities. Semantic description of cooperatives interfaces are published in the registry: the flow control, the data flow and a semantic description of cooperation activities. We have described activities using a language inspired by OWL-S [22].

Step 2: Workflow Interconnection. To carry out a work that is not within the range of only one organization, a partner begins by searching organizations with complementary skills via the cooperative interfaces they published (see the second block of the Figure 2). In order to construct a virtual organization, we have to match cooperative interfaces. Matchmaking takes into account the flow control, the data flow and semantic descriptions of cooperation activities. Given two cooperative interfaces, the matchmaking result can be (1) positive (*i.e.* interfaces match) (2) negative (*i.e.* interfaces do not match) or (3) conditional (*i.e.* interfaces match if a given condition holds). If the matchmaking result is not negative, the cooperative interfaces are then interconnected.

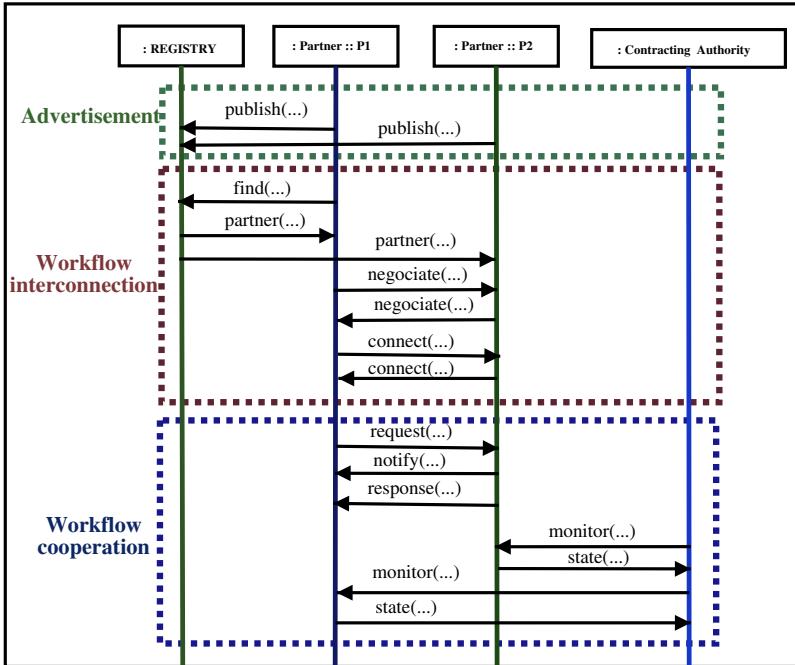


Fig. 2. Sequence diagram for workflow cooperation

In our example presented above, the matchmaking result is conditional: the client cooperative interface (see Figure 1-(b)) and the provider cooperative interface (see Figure 1-(d)) match if the client is a subscriber. Note that here we did not consider the semantic matchmaking. This is not within the scope of this paper. The result of this step (*i.e.* workflow interconnection) is an inter-organization workflow, presented in Figure 3, and a set of cooperation policies that define cooperative partners and their interfaces (*i.e.* cooperative activities, their order of execution,...) and constraints on workflow interactions (*e.g.* the matchmaking condition).

Step 3: Workflow Cooperation and Monitoring. The third and last step within our approach for workflow cooperation consists in the inter-organizational workflow deployment and execution. To do that, we have developed *CoopFlow*, a workflow cooperation framework, that allows different WfMSs to interconnect and cooperate their workflows. In addition to cooperation, this framework mainly enforces cooperation policies identified during the workflow interconnection step.

Cooperation policies enforcement could be implemented by many styles, we can cite: (1) organizations trust each other sufficiently to interact directly with each other. Cooperation policies enforcement is completely distributed. (2) no direct trust between the organizations exist, so interactions take place through trusted third parties acting as intermediaries.

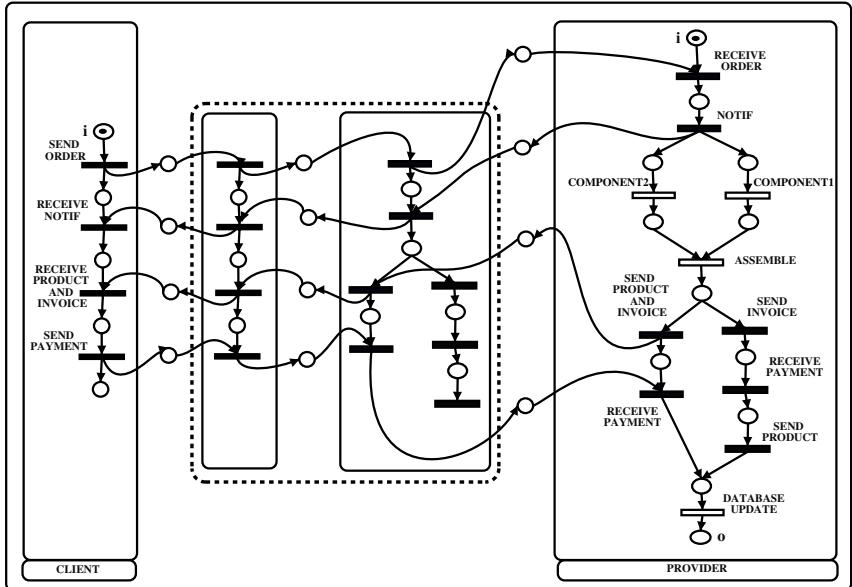


Fig. 3. Interconnection of client and provider workflows

In Figure 2, contracting authority is presented as one logical entity that can be implemented by one or several physical entities. In this later case, additional steps in the diagram are required as illustrated by Wombacher [23]. In the rest of this paper, we focus on the architecture of the framework we propose as well as its implementation.

4 The *CoopFlow* Architecture

After presenting the different steps of our workflow cooperation approach, we focus in the following, on the *CoopFlow* architecture.

One of the objectives of our framework is to allow partners distributed in time and place, to cooperate with each other and provide them with powerful and flexible ways permitting not only to simplify the inter-organizational workflow interoperability but also cooperation. To ensure these ends, the framework we propose, allows a dynamic join/leave of all existing WfMSs that are able to call external applications and that allow external applications to invoke any step within a workflow they manage. Moreover, our framework respects the privacy of participating organizations as well as their autonomy by reducing the workflows inter-visibility as tiny as the cooperation needs, based on the view principle. In addition, in order to enable cooperative organizations to integrate their disparate workflows, our framework fully integrates pre-established workflows. Furthermore, the framework allows the integration of WfMS's implementing standardized interoperability languages such as *Wf-XML*. Finally, the

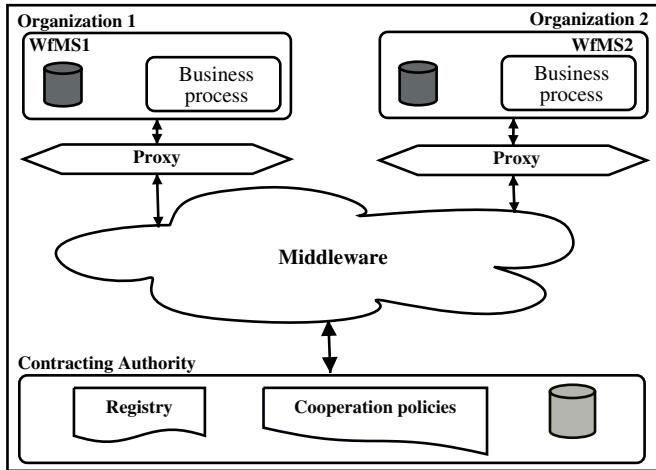


Fig. 4. *CoopFlow*: A framework for inter-organizational workflow

framework offers centralized and distributed control of cooperation depending on the trust degree between participating organizations. The global architecture is composed of a set of cooperating partners, a possible contracting authority (in the case where there is no sufficient trust between partners), and proxies. The communication between organizations is ensured by a middleware (see Figure 4).

In the remaining of this section, we detail the different components of *CoopFlow* in both centralized and distributed cases. Figure 5 illustrates the case of centralized control where there is no sufficient trust between partners. While Figure 6 shows the case of distributed control.

4.1 Contracting Authority

Once each participating organization identifies the different partners with complementary skills to cooperate with and cooperation policies are established, the different partners interconnect to each other and are ready to cooperate. The contracting authority role is to monitor and control the workflow cooperation. It includes a registry containing the partners' profiles, the cooperation policies defining the partners' responsibilities, a controller and a database (see Figure 5).

The **registry** provides enterprises with searching and publication capabilities to allow them to get partners with useful skills and giving them the ability to share business semantic information and business process resources. Moreover, registry technology enables trading partners to identify common data sets, data structures, business processes and mappings. Furthermore, the registry offers a matchmaking service that allows bringing together partners with complementary skills in order to construct an inter-organizational workflow. Matchmaking takes into account the flow control, the data flow and semantic descriptions of cooperation activities. Given two cooperative interfaces, the matchmaking result can be (1) positive (*i.e.* interfaces match) (2) negative (*i.e.* interfaces do not match)

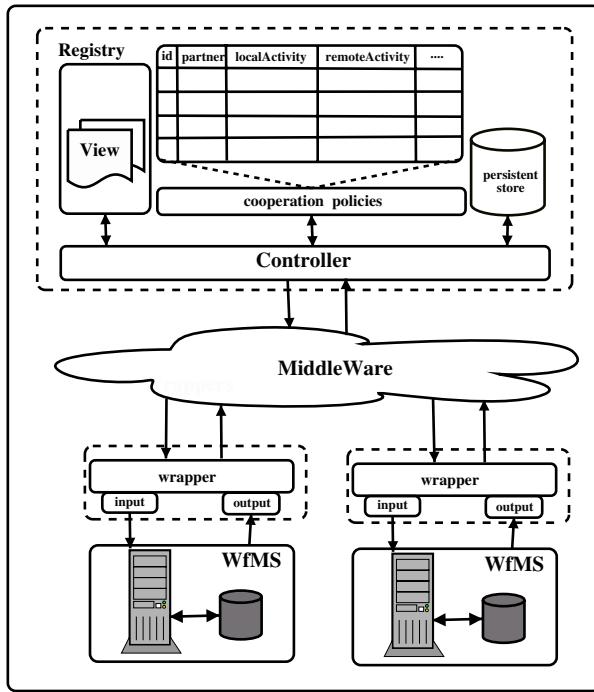


Fig. 5. *CoopFlow*: A centralized architecture

or (3) conditional (*i.e.* interfaces match if a given condition holds). If the matchmaking result is not negative, the cooperative interfaces are interconnected. The result of this matchmaking is a set of cooperative policies.

The **cooperation policies** describe the responsibilities and the roles played by the partners in the cooperation. They define cooperative partners and their interfaces (*i.e.* cooperative activities, their order of execution,...) and constraints on workflow interactions (*e.g.* the matchmaking condition).

The **local database** serves to store local and transient information during the execution of the inter-organizational workflow (*e.g.* workflow and cooperative execution state)

The **controller** represents the regulating body of the proxy that manages interactions between workflows according to the established cooperation policies. It is generic and independent of WfMSs of participating organizations. This component belongs to the contracting authority if no direct trust between the organizations exists, so interactions take place through trusted third parties acting as intermediaries. Otherwise, *i.e.* organizations trust each other sufficiently to interact directly with each other, cooperation policies enforcement is completely distributed and assumed by the controllers of the different proxies.

Figure 6 illustrates the case where organizations trust each other. In addition to the internal and external requests control and coordination, the controller permits to interact with the contracting authority as well as controllers of partners

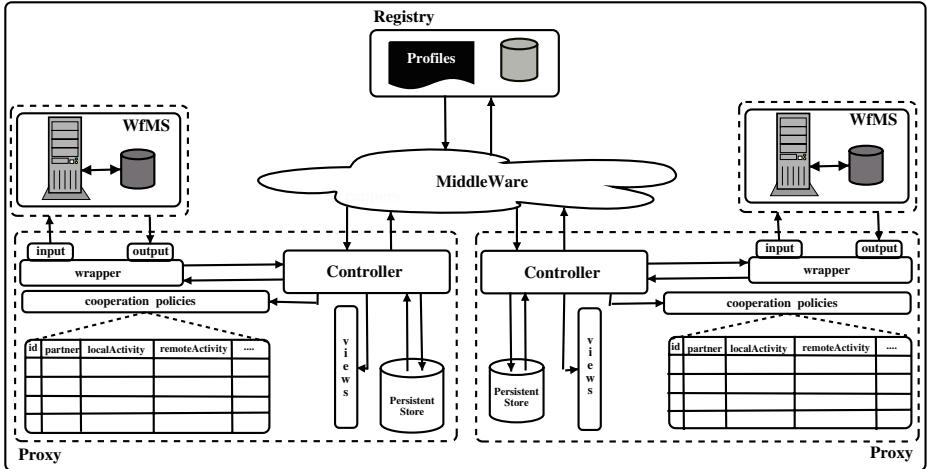


Fig. 6. *CoopFlow: A distributed architecture*

with complementary skills in the case of distributed architecture. In short, the controller permits functionalities: (1) partners profiles search and publication, (2) partners interconnection, (3) send/reception of data to/from partners, and (4) monitoring and control of the inter-organizational workflows cooperation.

When receiving an incoming request, the controller authenticates the sender and verifies whether it is allowed to execute the asked actions and/or transmit data. In the case where the requested operations are authorized, the controller forwards the request as well as the corresponding data to the WfMS via the wrapper. Otherwise, the controller rejects the sender request.

4.2 The Proxy Architecture

In the case where no direct trust between the organizations exists, interactions take place through trusted third parties acting as intermediaries that includes the cooperation policies. Otherwise, the interaction is direct and cooperation policies are distributed among partners and both monitoring and control are ensured by proxies.

In the following we present the second case where interactions are assumed by the partners' proxies. Each partner proxy is composed of a Controller, a Wrapper and Cooperation policies components. The controller is generic, *i.e.* independent of the plugged WfMS whereas the wrapper depends on how plugged WfMS interact with its environment. The organization of the proxy as generic and specific parts provides powerful ways for inter-organizational workflow cooperation and hard code reduction. Furthermore, this provides workflow participants with the freedom to change their specific workflows without changing the behaviour of the global proxy. This increases flexibility and is an important step to increase efficiency as well as reduction in costs for inter-organizational workflows and permit to easily change internal WfMS.

In the case of a distributed architecture, the controller interacts with other controllers of partners with complementary skills. The wrapper represents an internal component interfacing the specific WfMS with the controller. In fact, when receiving an incoming request (from the Controller or the WfMS), it adapts the call and then sends it to its destination (controller or WfMS). Two types of wrappers are offered by the framework: standardized wrappers that allow WfMS implementing some standards such as *Wf-XML* to directly plug-in into our framework without any added code. This kind of wrappers support operations allowing the WfMS interoperability and extends them with cooperation capabilities such monitoring and control features according to established cooperation policies. The second type of wrappers represents the specific wrappers that are implemented in the case where WfMS don't implement any interoperability language or use a non-standardized one. This allows them, to be interfaced with the controller and adapt specific WfMS operations to be able to communicate with the generic controller. Finally, cooperation policies determine the partners' roles and interactions rules as well as all data required for the cooperation accomplishment. These data include partner views, cooperative interfaces, partners responsibilities, . . . and are preserved in persistent stores.

5 Implementation Issues

Above, we have presented the proxy architecture and described its different components. In this section, we present how we have implemented the proxy and then we present the implementation of Proxies for *Xflow*, *Osworkflow*, and *Shark* WfMS.

5.1 Proxy Implementation

The proposed framework supports the dynamic cooperation between organizations as well as dynamic and transparent integration of existing WfMSs distributed in time and place. This permits not only to simplify the inter-organizational workflows cooperation but also to hide the complexity of lower-level and location details for participating partners. To ensure these objectives, the chosen implementation technology must define facilities not only for development and deployment of transactional and distributed object but also for creating composable components. Among the solutions proposed to satisfy the mentioned needs, we have chosen the Enterprise JavaBeans (EJB) for facilitating the re-use as well as interoperability of components and thus their development and their integration, and simplifying the transaction management. In the remaining of this section we present the Proxy different components implementation that are the *Controller*, the *wrapper*, *Cooperation Policies* and *Interconnection*. (see Figure 7).

Controller: The controller permits to implement a business activity allowing the send/reception of data as well as orders/notifications, and may be used by remote applications such as remote partners with complementary skills to

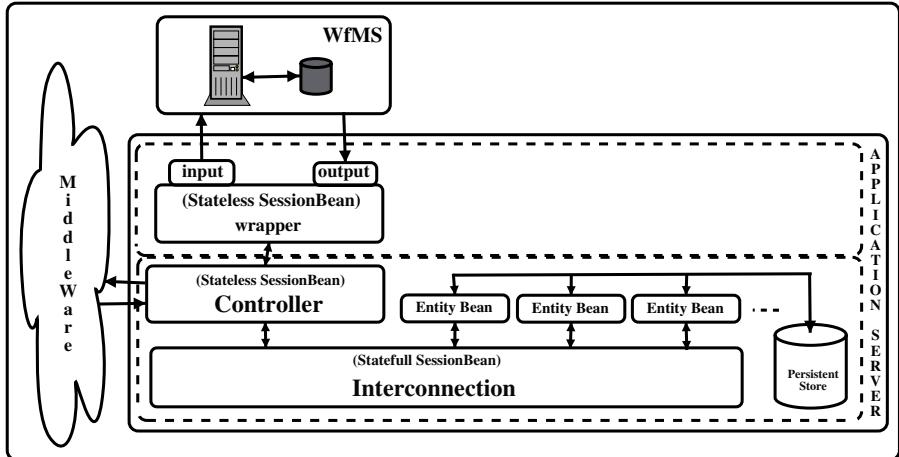


Fig. 7. Wrapper Implementation

cooperate with. Moreover, the controller is meant to be distributed and easy to access by hiding, from remote applications, its location as well as the complexity of its implementation details. Thus, it's well suited to expose the controller as a Web Service. Since *SOAP* and Web Services are stateless by nature, we use stateless session beans to expose the controller as Web Service.

Wrapper: The wrapper allows interfacing the controller with the WfMS and is implemented as a stateless session bean that is exposed as a Web Service. This allows it to be distributed and deployed in a location that can be different from the controller one.

Cooperation policies: After establishing contracts with partners having complementary skills, the controller needs to store cooperation policies and the different local data into databases. This information will be presented as business objects in a persistent storage mechanism using entity beans. Some examples of theses business objects are Rules, Partners, Views, Virtual Activities, and Mappings.

Interconnection: To conserve resources during the entire cooperation life, we have interfaced the controller and the different entity beans with a statefull session bean providing a set of methods permitting to the controller the manipulation of business objects. Hence, all data used and contracts required for the controller cooperation are persisted after the controller interrogation.

To allow the send and reception of requests/responses and orders/notifications, partners must just add a proxy composed of the components mentioned above: a controller stateless session bean, an interconnection statefull session bean and the set of entity beans allowing the resources preservation. To communicate with a specific WfMS, partners must add a stateless session bean to interface this later with the controller. To ensure internal interoperability between the controller

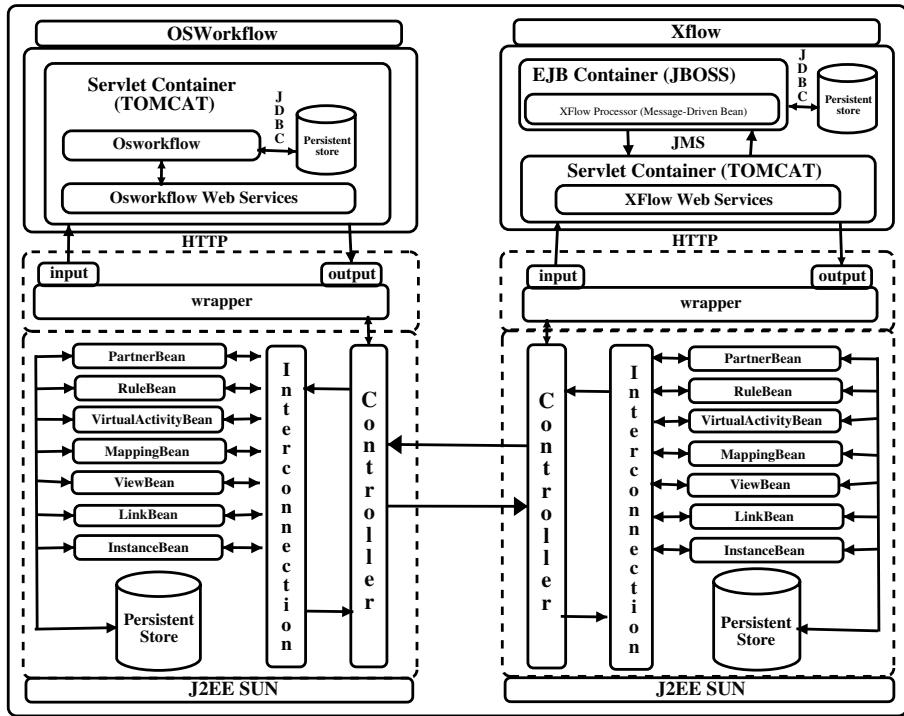


Fig. 8. Xflow and Oworkflow workflows cooperation

and the wrapper, the implementation we propose support organizations, either implementing *Wf-XML* or not. If the WfMS doesn't support *Wf-XML*, it must just add a wrapper to be interconnected to the controller.

5.2 Application : The Case of Xflow, Oworkflow and Shark

Recently, we have developed three proxies for three existing WfMSs: *XFlow*², *OSWorkflow*³, and *Shark*⁴ (see Figure 8). In the following, we present the three WfMS's and then describe the cooperation scenario between them.

XFlow is a pure J2EE framework for building, executing and managing business processes and workflows. It runs within an EJB and servlet container. JBoss 4.0 (with bundled Tomcat) is the container used in our implementation. The architecture of *XFlow* supports distributed and parallel processes within an organizations' firewall as well as across organizational boundaries.

OSWorkflow is a Java based open source project. In our application it runs within the servlet container Tomcat and communicates with database via JDBC.

² Xflow: <http://xflow.sourceforge.net/>

³ OSWorkflow: <http://www.opensymphony.com/osworkflow/>

⁴ Shark: <http://shark.objectweb.org/>

Shark is an extendable workflow engine framework including a standard implementation completely based on *WfMC* specifications using *XPDL* as its native workflow process definition format and the *WfMC* ToolAgents API for server side execution of system activities.

We mention here that all of these WfMSs fulfil cooperation conditions that are the capability of calling external applications and allowing these ones to invoke any step within a workflow they manage.

The communication between the systems is supported by a Web service middleware. In order to preserve the systems privacy and limit accesses to workflow resources (data access rights, method visibility, etc.), incoming/outgoing invocations are filtered by the proxies. For *Shark* interoperability, we have developed a *Wf-XML* wrapper. The wrapper presents the advantage of being generic and allows the plug-in of any WfMS implementing *Wf-XML* specification. In fact, if a cooperating WfMS implements the fourth interface of *WfMC*, then it would be directly connected to our framework without added code. Otherwise, organizations must just add a simple wrapper to interconnect and cooperate within our framework. The implemented proxies are in the form of EJBs deployed in J2EE application servers where both of the controller and the wrapper are exposed as Web services.

6 Conclusions

In this paper we have presented a framework enabling the cooperation of inter-organizational workflows. The framework we proposed allows the plug-in of any existing WfMS able to invoke external applications and that can be invoked from external applications (including workflows hosted by heterogeneous a WfMS). To adapt incoming and outgoing invocations and control them we have used proxies that are organized in term of a generic part that is common to all participating organizations and a specific one permitting the interaction with specific WfMS. This provides powerful ways for inter-organizational workflow cooperation and code reduction. In fact, this allows partners to change the WfMS executing their workflow without changing the global proxy behaviour and this separation permits to deploy the framework components in different locations.

Furthermore, the cooperation framework we propose provides a high degree of flexibility and addresses to workflows cooperation requirements. In fact, it permits a dynamic interconnection and disconnection of participating organizations. Moreover, it preserves the privacy and autonomy of process participants by reducing inter-visibility as tiny as the cooperation needs based on the view principle. In addition, our framework preserves established workflows. Participants don't modify their internal system. Instead, they have just to implement a proxy to integrate our framework.

To validate and test our framework applicability, we have built a prototype permitting the cooperation between three existing WfMS that are able to invoke external application and that can be invoked from external applications, *Xflow*, *Osworkflow* and *Shark*, by implementing the corresponding proxies. In addition,

we have developed a *Wf-XML* wrapper. The wrapper presents the advantage of being generic and allowing the plug-in of any WfMS implementing *Wf-XML* specification. In fact, if a cooperating WfMS implements the fourth interface of *WfMC*, then it would be directly connected to our framework without any added code. Otherwise, organizations must just add a simple wrapper to interconnect and cooperate within our framework.

References

1. Allen, R.: Workflow: An introduction. In Fisher, L., ed.: Workflow Handbook. Future Strategies, Lighthouse Point, FL (2001) 15–38
2. Tata, S., Boughzala, I.: Modeling contribution for virtual enterprise support. In: 12th IEEE International Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises, Linz, Austria, IEEE Computer Society (2003) 165–170
3. Tata, S., Chebbi, I.: A bottom-up approach to inter-enterprise business processes. In: 13th IEEE International Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises, Modena, Italy (2004)
4. Chebbi, I., Tata, S., Dustdar, S.: Cooperation policies for inter-organizational workflows. In: Teamware: supporting scalable virtual teams in multi-organizational settings, Symposium on Applications and the Internet, Trento, Italy (2005)
5. Chebbi, I., Dustdar, S., Tata, S.: The view-based approach to dynamic inter-organizational workflow cooperation. Data and Knowledge Engineering Journal (2005, forthcoming)
6. Griffel, F., Boger, M., Weinreich, H., Lamersdorf, W., Merz, M.: Electronic contracting with cosmos - how to establish, negotiate and execute electronic contracts on the internet. In: Enterprise Distributed Object Computing Workshop. (1998)
7. Verharen, E., Papazoglou, M.: Introduction to contracting in distributed transactional workflow. In: Annual Hawaii International Conference on System Science. (1998)
8. Grefen, P., Aberer, K., Hoffer, Y., Ludwig, H.: Crossflow: Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. IEEE Data Engineering Bulletin **24** (2001) 52–57
9. Alonso, G., Fiedler, U., Hagen, C., Lazcano, A., Schuldt, H., Weiler, N.: Wise: Business to business e-commerce. International Workshop on Research Issues on Data Engineering: Information Technology for virtual Enterprises (1999)
10. Lazcano, A., Alonso, G., Schuldt, H., Schuler, C.: The wise approach to electronic commerce. International Journal of Computer Systems Science & Engineering, special issue on Flexible Workflow Technology Driving the Networked Economy **vol. 15** (2000)
11. Kotok, A., Webber, D.R.: ebXML: The New Global Standard for Doing Business on the Internet. New Riders Publishing (2001)
12. Christian, B.H.: ebxml: Status, research issues, and obstacles (2002)
13. van der Aalst, W.M.P.: Loosely coupled interorganizational workflows: Modeling and analyzing workflows crossing organizational boundaries. Information and Management **37** (2000) 67–75
14. van der Aalst, W.M.P.: Process-oriented architectures for electronic commerce and interorganizational workflow. Inf. Syst. **24** (1999) 639–671
15. Muth, P., Wodtke, D., Weissfels, J., Angelika Kotz Dittrich, G.W.: From centralized workflow specification to distributed workflow execution. Journal of Intelligent Information Systems (1998) 10(2):159–184

16. van der Aalst, W.M.P., Weske, M.: The p2p approach to interorganizational workflows. In: Proceedings of the 13th International Conference on Advanced Information Systems Engineering, Springer-Verlag (2001) 140–156
17. Zhao, J.L.: Workflow management in the age of e-business. In: Tutorial at the 35th Hawaii International Conference on System Sciences, Waikoloa, Hawaii (2002)
18. Hollingsworth, D.: An xml based architecture for collaborative process management. In Fischer, L., ed.: Workflow Handbook 2002. Future Strategies, Lighthouse Point (FL) (2002) 95–116
19. WfMC: Workflow process definition interface - xml process definition language. document status - 1.0 final draft. Document number wfmc-tc-1025, Workflow Management Coalition (2002)
20. Ricker, J., Swenson, K.D., Silverstein, M.: Asynchronous service access protocol working draft 01. Standards specification, Organization for the Advancement of Structured Information Standards (2003)
21. van der Aalst, W.M.P.: Interorganizational workflows: An approach based on message sequence charts and petri nets. *Systems Analysis - Modelling - Simulation* **34** (1999) 335–367
22. Martin, D.L., Paolucci, M., McIlraith, S.A., Burstein, M.H., McDermott, D.V., McGuinness, D.L., Parsia, B., Payne, T.R., Sabou, M., Solanki, M., Srinivasan, N., Sycara, K.P.: Bringing semantics to web services: The owl-s approach. In: Semantic Web Services and Web Process Composition Workshop. Volume 3387 of Lecture Notes in Computer Science., San Diego, CA, USA (2004) 26–42
23. Wombacher, A., Fankhauser, P., Aberer, K.: Overview on decentralized establishment of consistent multi-lateral collaborations based on asynchronous communication. In: IEEE International Conference on e-Technology, e-Commerce, and e-Services, IEEE Computer Society (2005) 164–170

Process Mining and Verification of Properties: An Approach Based on Temporal Logic

W.M.P. van der Aalst, H.T. de Beer, and B.F. van Dongen

Department of Technology Management,
Eindhoven University of Technology, P.O.Box 513,
NL-5600 MB, Eindhoven, The Netherlands
`w.m.p.v.d.aalst@tm.tue.nl`

Abstract. Information systems are facing conflicting requirements. On the one hand, systems need to be adaptive and self-managing to deal with rapidly changing circumstances. On the other hand, legislation such as the Sarbanes-Oxley Act, is putting increasing demands on monitoring activities and processes. As processes and systems become more flexible, both the need for, and the complexity of monitoring increases. Our earlier work on *process mining* has primarily focused on *process discovery*, i.e., automatically constructing models describing knowledge extracted from event logs. In this paper, we focus on a different problem complementing process discovery. Given an event log and some property, we want to *verify* whether the property holds. For this purpose we have developed a new language based on Linear Temporal Logic (LTL) and we combine this with a standard XML format to store event logs. Given an event log and an LTL property, our LTL Checker verifies whether the *observed behavior* matches the *(un)expected/(un)desirable behavior*.

Keywords: Process mining, temporal logic, business process management, workflow management, data mining, Petri nets.

1 Introduction

A constantly changing reality is forcing organizations and their information systems to adapt at an ever increasing pace. Business Process Management (BPM) and Workflow Management (WFM) systems increasingly allow for more flexibility. Instead of recoding the system it typically suffices to reconfigure the system on the basis of a process model [3]. Several researchers have addressed the problems related to workflow change [1,15,29,30]. Although the work on workflow change is highly relevant, in reality many processes are not bound by a WFM system, or a BPM system driven by an explicit process model. In contrast, some systems, e.g., the case handling system FLOWer, allow for implicit routing, other systems allow for much more behavior than desired. For example, people using the SAP R/3 system are not limited by process models described in the SAP R/3 Reference Model database [25]. Deviations from the “normal process” may be desirable but may also point to inefficiencies or even fraud. New legislation

such as the Sarbanes-Oxley (SOX) Act [33] and increased emphasis on corporate governance has triggered the need for improved auditing systems [23]. For example, Section 404 of the SOX Act [33] states two requirements: (1) Section 404(a) describes management's responsibility for establishing and maintaining an adequate internal control structure and procedures for financial reporting and assessing the effectiveness of internal control over financial reporting, and (2) Section 404(b) describes the independent auditors responsibility for attesting to, and reporting on, management's internal control assessment. Both requirements suggest an increased need for the detailed auditing of business activities. To audit an organization, these business activities need to be monitored. As enterprises become increasingly automated, a tight coupling between auditing systems and the information systems supporting the operational processes becomes more important.

Today's information systems need to compromise between two requirements: (1) being adaptive and self-managing and (2) being able to be audited. Within the context of this struggle, we have developed a tool called *LTL Checker*. This tool has been developed in the context of our *ProM framework*¹. The ProM framework offers a wide range of tools related to process mining, i.e., extracting information from event logs [7]. Process mining is motivated by the fact that many business processes leave their "footprints" in transactional information systems (cf. WFM, ERP, CRM, SCM, and B2B systems), i.e., business events are recorded in so-called event logs. Until recently, the information in these logs was rarely used to analyze the underlying processes. Process mining aims at improving this by providing techniques and tools for discovering process, control, data, organizational, and social structures from event logs, i.e., the basic idea of process mining is to diagnose business processes by mining event logs for knowledge.

The work presented in this paper is related to process mining, but, unlike most process-mining approaches, the emphasis is not on discovery. Instead we focus on *verification*, i.e., given an event log we want to verify certain properties. One example is the *4-eyes principle*, i.e., although authorized to execute two activities, a person is not allowed to execute both activities for the same case. For example, a manager may submit a request (e.g., to purchase equipment, to make a trip, or to work overtime) and he may also approve requests. However, it may be desirable to apply the 4-eyes principle implying that the manager is not allowed to approve his own request. If there is an event log recording the events "submit request" and "approve request", the 4-eyes principle can be verified easily. More difficult are properties relating to the ordering or presence of activities. For example, activity *A* may only occur if it is preceded by activity *B* or activity *C* and immediately followed by activity *D*. Therefore, we propose an approach based on *temporal logic* [26,28]. More specifically, we use an extension of *Linear Temporal Logic* (LTL) [17,20,21] tailored towards event logs holding information on activities, cases (i.e., process instances), timestamps, originators (the person or resource executing the activity), and related data.

¹ Both documentation and software can be downloaded from www.processmining.org.

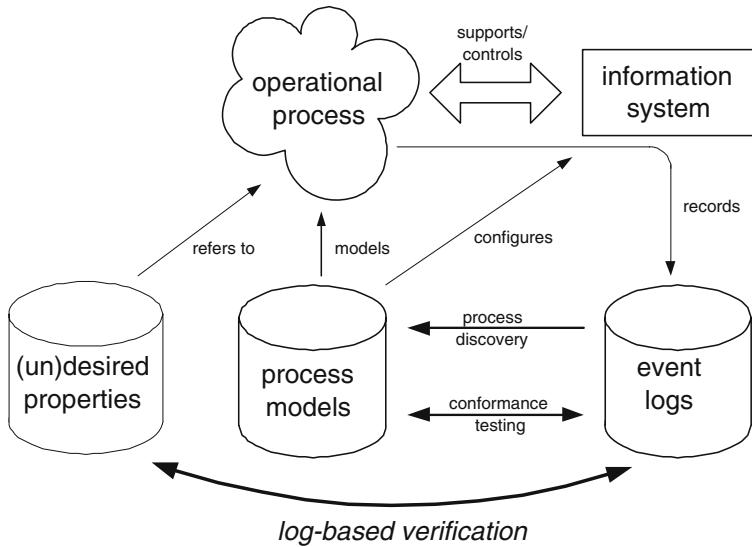


Fig. 1. The role of verification in the context of process mining

Figure 1 shows the “bigger picture”, i.e., the role of log-based verification in relation to other process-mining methods such as process discovery. The information system is there to support, control and/or follow operational processes. Through the information system (parts of) the operational process is (are) recorded. Using process discovery, models (e.g., Petri nets) may be generated to explain the observed behavior. Unfortunately, process discovery may be intractable for many processes. However, for most processes one can formulate (un)expected/(un)desirable properties. These properties can be directly compared with the event log. Note that, in principle, one can first build a model and then compare the model and some property using model checking. We do not do this because in the discovery process typically information is lost and for some logs process discovery may be intractable. Note that log-based verification is also different from conformance testing where the model and event log are compared (see Figure 1). Log-based verification is more robust than most other process-mining approaches. For example, even for very complicated processes resulting in spaghetti-like diagrams it is easy to verify the 4-eyes principle mentioned before.

This paper reports on the language developed to formulate properties in the context of event logs, the approach used to check these properties, the implementation of the LTL Checker in the ProM framework, and the relation between this work and process discovery. It is important to note that process discovery is difficult in situations where a lot of flexibility is offered. As indicated, an approach based on verification is more robust because it can focus on the essential properties. Hence, the LTL Checker is a welcome addition towards a wider range of process mining tools.

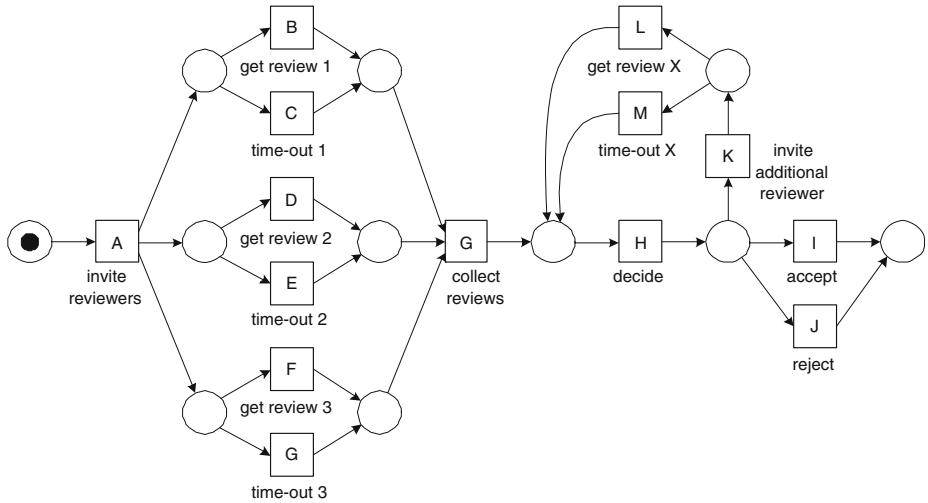
This paper is organized as follows. Section 2 introduces a running example that will be used to illustrate the concept of process mining. The ProM framework and the XML format used to store event logs is presented in Section 3. Then the new LTL language is introduced and it is shown how properties can be specified. Section 5 shows how these properties can be verified using the newly developed LTL Checker in ProM. Finally, some related work is discussed and the paper is concluded.

2 Running Example

Today, many enterprise information systems store relevant events in some structured form. For example, WFM systems typically register the start and completion of activities. ERP systems like SAP log all transactions, e.g., users filling out forms, changing documents, etc. Business-to-Business (B2B) systems log the exchange of messages with other parties. Call center packages but also general-purpose Customer Relationship Management (CRM) systems log interactions with customers. These examples show that many systems have some kind of *event log* often referred to as “history”, “audit trail”, “transaction log”, etc. [7,8]. The event log typically contains information about events referring to an *activity* and a *case*. The case (also named process instance) is the “thing” which is being handled, e.g., a customer order, a job application, an insurance claim, a building permit, etc. The activity (also named task, operation, action, or work-item) is some operation on the case. Typically, events have a *timestamp* indicating the time of occurrence. Moreover, when people are involved, event logs will typically contain information on the person executing or initiating the event, i.e., the *originator*. Based on this information several tools and techniques for process mining have been developed.

As a running example, we will use the process shown in Figure 2. The process describes the reviewing process of a paper for a journal and is represented in terms of a Petri net (more specifically a workflow net [3]). After inviting three reviewers (activity *A*) each of the reviewers returns a review or a time-out occurs, e.g., for the first review either *B* or *C* occurs. Then the reviews that were returned in time are collected (activity *G*) and a decision is made (activity *H*). There are three possible outcomes of this decision: (1) the paper is accepted (*I*), (2) the paper is rejected (*J*), or (3) an additional reviewer is needed (*K*). Similar to the original three reviewers, the additional reviewer may return the review in time (*L*) or not (*M*).

For the process shown in Figure 2, we may log events such as the ones shown in Table 1. As discussed before, each event refers to a case (e.g., *case_1*) and an activity (e.g., invite reviewers). Moreover, in this case the timestamp and originator are logged. The first line of the fragment shown in Table 1 states that John executed step *A* (invite reviewers) for *case_0* on the first of January 2005. Table 1 does not show that for some events additional data is logged. For example, each case has a data element *title* and each review result (e.g., get review 1) has a *result* attribute which is either accept or reject. Table 1

**Fig. 2.** Running example**Table 1.** A fragment of the event log

case id	activity id	originator	timestamp
...			
case_0	invite reviewers	John	2005-01-01T08:00
case_1	invite reviewers	John	2005-01-01T08:00
case_0	get review 1	Nick	2005-02-06T08:00
case_0	get review 2	Pete	2005-03-07T08:00
...			

only shows a fragment of the log used throughout this paper. The log holds information about 48 cases (i.e., papers) and 354 events and is used as a running example.

3 ProM Framework and XML Format

The LTL Checker presented in this paper is embedded in the ProM framework and should be seen as an addition to a collection of process mining tools. Therefore, we first describe the ProM framework and some of the process mining techniques that have been developed in this framework. The goal of this section is to introduce the format used to log events and to provide a brief overview of some of the techniques complementing the results presented in this paper.

In Table 1 we showed a fragment of some event log. We assume a standard log format, named MXML, and have developed several adaptors to map logs in different information systems onto our log format (e.g., Staffware, FLOWER, MS Exchange, MQSeries, etc.). Figure 3 shows the hierarchical structure of

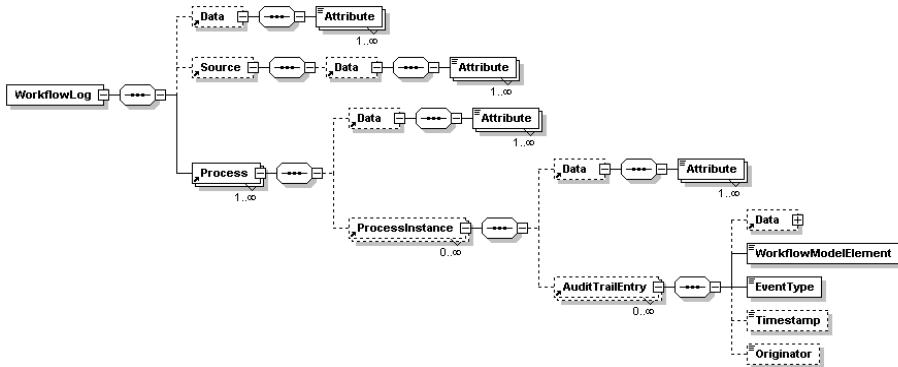


Fig. 3. XML schema for the MXML format used by ProM

MXML. The format is XML based and is defined by an XML schema (cf. www.processmining.org).

The ProM framework has been developed as a completely plug-able environment. It can be extended by simply adding plug-ins, i.e., there is no need to know or recompile the source code. Currently, more than 30 plug-ins have been added. The most interesting plug-ins are the mining plug-ins and the analysis plug-ins. The architecture of ProM allows for five different types of plug-ins:

Mining plug-ins which implement some mining algorithm, e.g., mining algorithms that construct a Petri net based on some event log.

Export plug-ins which implement some “save as” functionality for some objects (such as graphs). For example, there are plug-ins to save EPCs, Petri nets, spreadsheets, etc.

Import plug-ins which implement an “open” functionality for exported objects, e.g., load instance-EPCs from ARIS PPM.

Analysis plug-ins which typically implement some property analysis on some mining result. For example, for Petri nets there is a plug-in which constructs place invariants, transition invariants, and a coverability graph.

Conversion plug-ins which implement conversions between different data formats, e.g., from EPCs to Petri nets.

For the process perspective, four mining plug-ins are available including the α plug-in [9], a genetic miner, and a multi-phase miner. The goal of these plug-ins is to extract a process model from a given event log without using any additional knowledge of the process. For example, based on the log mentioned in Section 2 (i.e., the log holding information on 48 papers and 354 events), the α plug-in discovers the process model shown in Figure 4. Note that this model is identical to the one shown in Figure 2. Of course the layout is different since it is automatically generated. For the organizational/social perspective, one mining plug-in named MinSoN is available [6]. If we apply this plug-in to the same log, ProM constructs the social network shown in Figure 5. A social network shows all actors in the organization and their relationships. Based on an analysis of the log

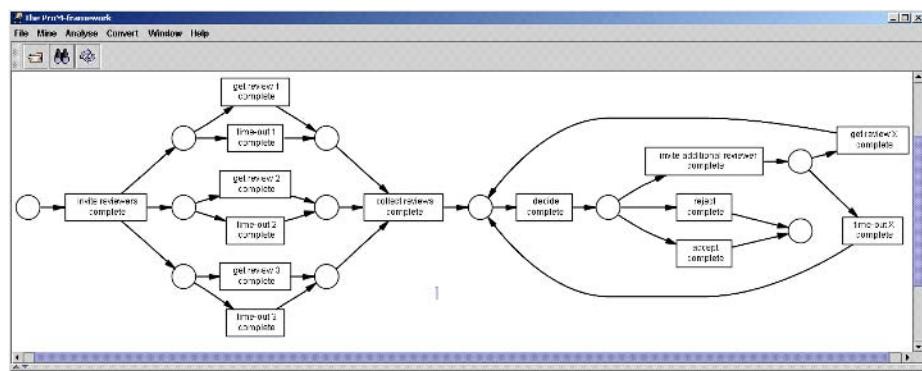


Fig. 4. The α mining plug-in in ProM used to discover the underlying process model

(e.g., transfer of work or similarity of work profiles), the relationships and their relative strength are derived. Figure 5 shows how these can be analyzed, e.g., using a tool like NetMiner. The screenshot on the left shows that John and Mike are the two central players in the reviewing process. This is no surprise since John is the editorial assistant (responsible for the contacts with reviewers and authors) and Mike is the editor of the journal. The screenshot on the right-hand-side of Figure 5 illustrates how NetMiner can discover “cliques” in a social network.

Figures 4 and 5 show how process mining techniques can be used to discover models based on some event log. The results presented in this paper are related to process mining, but unlike the mining plug-ins mentioned the focus is not on discovery. Instead, the focus is on verification. Therefore, in the context of the ProM framework, the LTL Checker should be considered as an analysis plug-in rather than a mining plug-in.

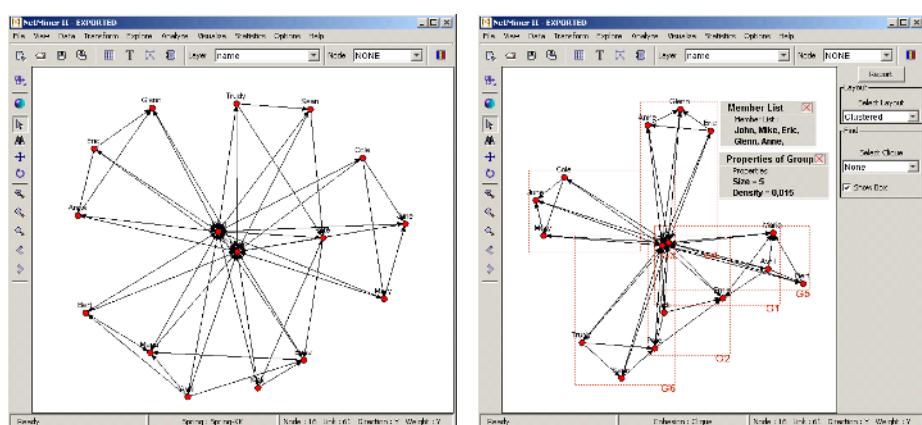


Fig. 5. The social network discovered by ProM exported to the SNA tool NetMiner

4 Formulating Properties: The LTL Language

Assuming that the information system at hand left a “footprint” in some event log, it is interesting to check whether certain properties hold or not. Before being able to check such properties, a concrete language for formulating dynamic properties is needed. Given the fact that we consider behavioral properties where ordering and timing are relevant, some temporal logic seems to be the best basis to start from [26,28]. There are two likely candidates: Computational Tree Logic (CTL) and Linear Temporal Logic (LTL) [17,20,21]. Given the linear nature of an event log, LTL is the obvious choice. It would only make sense to use CTL if first a model (e.g., an automaton) was built before evaluating the property. Unlike most of the existing process mining techniques supported in the ProM framework, we try to avoid this and simply use LTL as a basis directly working on the event log.

It is not sufficient to select LTL as a language. To easily specify properties in the context of MXML, a dedicated language is needed that exploits the structure shown in Figure 3. Therefore, in addition to standard logical operators, we need dedicated statements to address the properties of cases and events. For example, it should be easy to use the various attributes of a case (both standard ones such as case, activity, timestamp, originator and event type, and context specific ones such as data values).

We have developed a powerful language that includes type definitions, renaming, formulas, subformulas, regular expressions, date expressions, propositional logic, universal and existential quantification, and temporal operators such as nexttime ($\circ F$), eventually ($\diamond F$), always ($\Box F$), and until ($F \sqcup G$). A complete description of the language is beyond the scope of this paper but is given in [11]. To illustrate the language we use the examples shown in Table 2.

The notation `ate.X` is used to refer to some attribute `X` of an audit trail entry (`ate`), i.e., an event in the event log. Similarly `pi.X` is used to refer to attribute `X` of a process instance (`pi`), i.e., a case. There are several predefined attributes, e.g., `ate.WorkflowModelElement` refers to the activity (or other process elements) being executed. `ate.Originator` is the resource executing it, i.e., the person. `ate.Timestamp` is the timestamp of the event. `ate.EventType` is the type of the event (i.e., schedule, assign, reassign, withdraw, start, complete, suspend, etc.). The three `set` declarations shown in Table 2 (lines 1-3) declare that `ate.WorkflowModelElement`, `ate.Originator`, and `ate.EventType` can be used for quantification, e.g., `ate.WorkflowModelElement` may refer to the activity related to the current event but may also be used to range over all activities appearing the a case. Line 5 declares the `DateTime` format used when specifying a value (note that this allows for shorter notations than the standard XML format). Line 6 declares a data attribute at the level of an event. The data attribute `result` is used to record the outcome of a single review in the running example. Line 7 shows a data attribute at the level of a case. Note that both attributes are of type `string`. To allow for shorter/customized names our language allows for renaming. As shown in lines 9-11, `ate.Originator`, `ate.Timestamp`, and `ate.WorkflowModelElement` are renamed to `person`, `timestamp`, and `activity` respectively. These names are used in the remainder of Table 2.

Table 2. Some LTL formulas for the running example

```

1 set ate.WorkflowModelElement;
2 set ate.Originator;
3 set ate.EventType;
4
5 date ate.Timestamp := "yyyy-MM-dd";
6 string ate.result;
7 string pi.title;
8
9 rename ate.Originator as person;
10 rename ate.Timestamp as timestamp;
11 rename ate.WorkflowModelElement as activity;
12
13 formula accept_or_reject_but_not_both() :=
14  (<>(activity == "accept") <-> !(<>(activity == "reject")));
15
16 formula action_follows_decision() :=
17  []( (activity == "decide" -> _0( ((activity == "accept" \/
18 activity == "reject") \/ activity == "invite additional reviewer") )));
19
20 subformula execute( p : person, a : activity ) :=
21  <> ( (activity == a \/ person == p) );
22
23 formula not_the_same_reviewer() :=
24  forall[p:person |
25    (((!(execute(p,"get review 1")) \/ !(execute(p,"get review 2")))) \/
26    (!!(execute(p,"get review 1")) \/ !(execute(p,"get review 3")))) \/
27    (!!(execute(p,"get review 2")) \/ !(execute(p,"get review 3")))];
28
29 subformula accept(a : activity ) :=
30  <> ( (activity == a \/ ate.result == "accept" ) );
31
32 formula dont Reject_paper_unjustified() :=
33  (((accept("get review 1") \/ accept("get review 2")) \/
34 accept("get review 3"))
35 -> <> ( activity == "accept" ) );
36
37 formula started_before_finished_after(start_time:timestamp,
38 end_time:timestamp) :=
39  (<>( timestamp < start_time ) \/ <>( timestamp > end_time ));

```

The goal of the LTL language is to specify properties. Properties are described using the **formula** construct. Formulas may be nested and can have parameters. To hide formulas that are only used indirectly, the **subformula** construct should be used. Table 2 describes five formulas and two subformulas. Lines 13-14 specify a formula without any parameters. The property holds for a given

event log if for each paper there was an acceptance (activity I in Figure 2) or a rejection (activity J in Figure 2) but not both. To formulate this both temporal and logical operators are used: $\langle\rangle$ is the syntax for the temporal operator eventually ($\diamond F$), $\langle\rangle$ denotes “if and only if”, and $!$ is the negation. Line 14 uses the shorthand `activity` defined in Line 11 twice. `activity == "accept"` is true if the `WorkflowModelElement` attribute of the current event points to the acceptance activity. Hence, $\langle\rangle(\text{activity} == \text{"accept"})$ holds if the acceptance activity was executed. Similarly, $\langle\rangle(\text{activity} == \text{"reject"})$ holds if the rejection activity was executed. Using $\langle\rangle$ and $!$ we can formulate that exactly one of these two should hold. The formula `accept_or_reject_but_not_both` can be evaluated for each case in the log. If it holds for all cases, it holds for the entire log.

Lines 16-18 define the property that any decision (activity H in Figure 2) should be *directly* followed by a rejection (J), acceptance (I) or invitation (K). The following logical and temporal operators are used to achieve this: $[]$ to denote the always operator ($\square F$), \rightarrow for implication, $_0$ denote the nexttime operator ($\circ F$), and \vee for the logical or. The part $[] (\text{activity} == \text{"decide"}) \rightarrow$ states that it should always be the case that if the current activity is decide, the following should hold. The second part starts with $_0$ to indicate that immediately after the decision step the current activity should be of one of the three mentioned.

The formula specified in lines 23-27 uses the parameterized subformula defined in lines 20-21. The subformula states whether at some point in time activity a was executed by person p . Note that both parameters are typed through the declarations in the top part of Table 2. Formula `not_the_same_reviewer` calls the subformula six times to express the requirement that no reviewer should review the same paper twice. In terms of Figure 2: activities B , D , and F should be executed by different people. Note that universal quantification over the set people involved is used (cf. `forall[p:person | ...]`) where `person` is renamed in Line 9 and declared to be a set type in Line 2.

The formula specified in lines 32-34 uses the parameterized subformula defined in lines 29-30. The subformula checks whether there is some event corresponding to activity a that has a data attribute `result` set to value accept, i.e., `ate.result == "accept"`. Note that `ate.result` was declared in Line 6. Formula `dont_reject_paper_unjustified` states that a paper with three positive reviews (three accepts) should be accepted for the journal.

The last formula in Table 2 (lines 36-37) shows that it is also possible to use timestamps. The formula has two parameters (start and end time) and it holds if each case was started before the given start time and ended after the given end time.

The formulas shown in Table 2 are specific for the running example introduced in Section 2. However, many generic properties can be defined, e.g., the *4-eyes principle*. Recall that this principle states that, although authorized to execute two activities, a person is not allowed to execute both activities for the same case. The following formula can be used to verify this:

```
formula four_eyes_principle(a1:activity,a2:activity) :=
forall[p:person |(!(execute(p,a1)) \vee !(execute(p,a2)))];
```

The property `four_eyes_principle("invite reviewers","decide")` checks whether activities A and H in Figure 2 are indeed executed by different people. This example and the formulas in Table 2 provide an impression of the LTL language we have developed. It can be seen as a temporal logic tailored towards events logs. For more details we refer to [11] and www.processmining.org. The next section elaborates on the tool support for this language.

5 Verifying Properties: The LTL Checker

In Section 3, the ProM framework has been introduced. To be able to verify properties using the language presented, three plug-ins are needed: (1) a mining plug-in to load and filter an MXML file, (2) an import plug-in to load LTL files like the one shown in Table 2, and (3) an analysis plug-in providing the graphical interface and doing the actual verification. For convenience a large number of generic properties have been specified (e.g., the 4-eyes principle). There are about 60 application-independent properties focusing on the `ate.WorkflowModelElement` (activity), `ate.Originator` (person), `ate.Timestamp`, and `ate.EventType` attributes. Only for specific questions (e.g., related to data) the user needs to specify new formulas. The 60 standard formulas are stored in a default file that can be applied directly without any knowledge of the LTL syntax described in the previous section. It is possible to link HTML markup to any formula. This markup is shown to the user when selecting a formula. This should support the selection and interpretation of the corresponding property. Note that formulas may be parameterized and users need to type a value for each parameter, e.g., the two activity names in `four_eyes_principle("invite reviewers","decide")`. The graphical user interface shows the HTML text and a form that need to be filled out, cf. Figure 6.

The implementation of the actual LTL Checker is rather complicated. However, the core structure of the checker is fairly straightforward as is sketched in the remainder of this section. Let L denote the event log and F a formula expressed in the language described in the previous section. (If F is parameterized, then it is evaluated for concrete parameter values.) $check_{log}(L, F) = \forall_{\pi \in L}(check(F, \pi, 0))$ evaluates to true if F holds for the log L . $\pi \in L$ is a process instance (i.e., case) represented by a sequence of audit trail entries (i.e., events). $|\pi|$ is the length of π , i.e., the number of events, and π_i is the $(i - 1)$ -th entry, i.e., $\pi = \pi_0\pi_1\dots\pi_{|\pi|-1}$.

$check(F, \pi, 0)$ checks whether the formula F holds for the first process-instance $\pi \in L$ (i.e., the π at position 0 in L). For temporal operators, the position in the sequence π is relevant as well. Therefore, let F denote a formula, π a case, and i a number ($0 \leq i < |\pi|$).

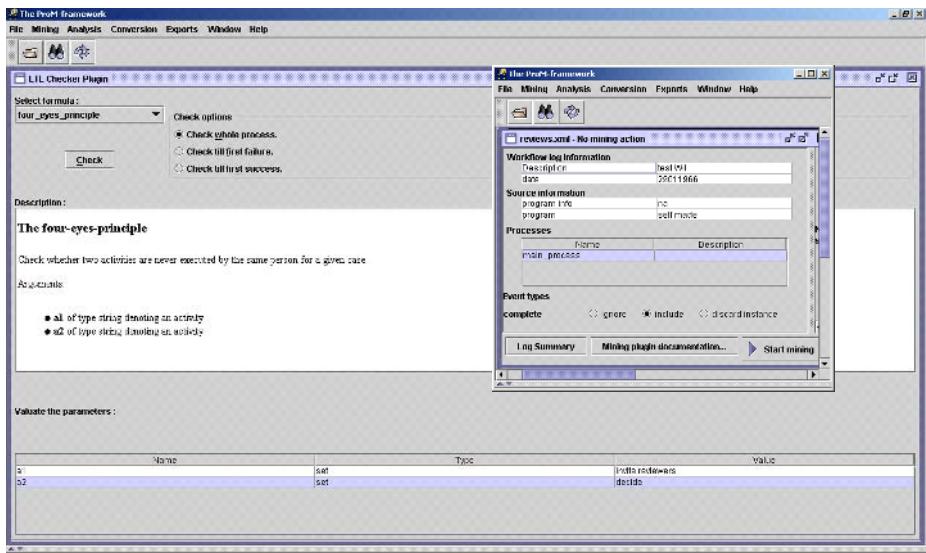


Fig. 6. Formula selection, help text, and configuration in the LTL Checker

$check(F, \pi, i) =$

if $F =$

$expr(\pi_i) \Leftrightarrow$

true, $expr$ is atomic and holds for i -th audit trail entry of π , i.e., π_i ;

false, $expr$ is atomic and does not hold for i -th audit trail entry of π ;

$\neg\phi \Leftrightarrow \neg check(\phi, \pi, i)$;

$\phi \wedge \psi \Leftrightarrow check(\phi, \pi, i) \wedge check(\psi, \pi, i)$;

$\phi \vee \psi \Leftrightarrow check(\phi, \pi, i) \vee check(\psi, \pi, i)$;

$\phi \rightarrow \psi \Leftrightarrow check(\phi, \pi, i) \rightarrow check(\psi, \pi, i)$;

$\phi \leftrightarrow \psi \Leftrightarrow check(\phi, \pi, i) \leftrightarrow check(\psi, \pi, i)$;

$\forall_{x \in X} (\phi_x) \Leftrightarrow \forall_{x \in X} (check(\phi_x, \pi, i))$;

$\exists_{x \in X} (\phi_x) \Leftrightarrow \exists_{x \in X} (check(\phi_x, \pi, i))$;

$\square\phi \Leftrightarrow$

$check(\phi, \pi, i) \wedge check(F, \pi, i + 1), 0 \leq i < (|\pi| - 1)$;

$check(\phi, \pi, i), i = (|\pi| - 1)$;

$\diamond\phi \Leftrightarrow$

$check(\phi, \pi, i) \vee check(F, \pi, i + 1), 0 \leq i < (|\pi| - 1)$;

$check(\phi, \pi, i), i = (|\pi| - 1)$;

$\circlearrowleft\phi \Leftrightarrow$

$check(\phi, \pi, i + 1), 0 \leq i < (|\pi| - 1)$;

$false, i = (|\pi| - 1)$;

$\phi \sqcup \psi \Leftrightarrow$

$check(\psi, \pi, i) \vee (check(\phi, \pi, i) \wedge check(F, \pi, i + 1)), 0 \leq i < (|\pi| - 1)$;

$check(\psi, \pi, i) \vee check(\phi, \pi, i), i = (|\pi| - 1)$;

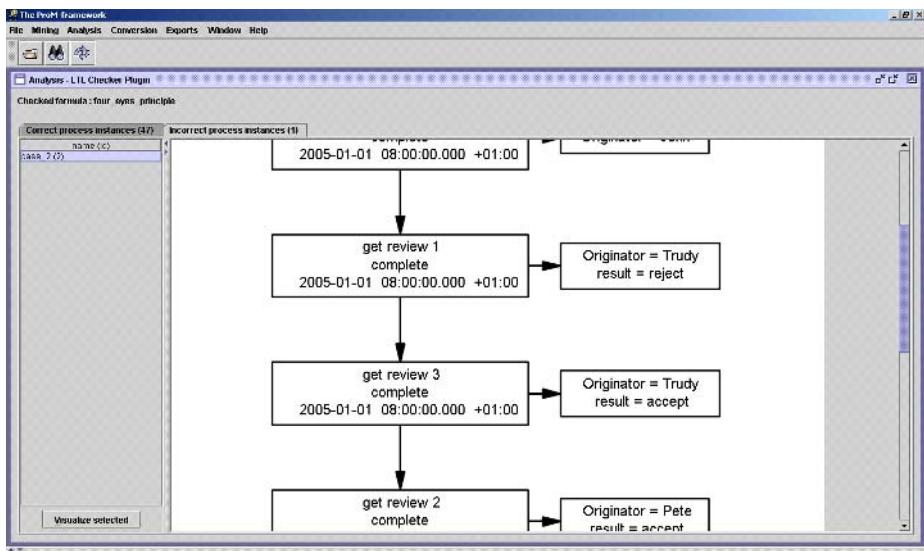


Fig. 7. The LTL Checker detected a problem when checking the 4-eyes principle

The *expr* function is a function which computes atomic Boolean expressions that may involve all kinds of attributes (e.g., timestamps etc. but also data values or case attributes). Given the fact that there are many comparison operators, typing issues and advanced features such as pattern matching, the coding of the LTL Checker is more complex than the sketch just given suggests.

The unfolding of the quantifications may be an expensive operation. However, no quantification is bigger than the number of events within a single case. Moreover, each case (process instance) can be checked in isolation thus making the algorithms tractable. Note that logs may contain thousands or even millions of cases. However, the number of events per case is typically less than 100. Therefore, from a practical point of view, the core algorithm is linear in the size of the log.

To conclude, we show a screenshot of the result, cf. Figure 7. It shows the result of `four_eyes_principle("get review 1","get review 3")` applied to the log with 48 cases. 47 of these cases satisfy the property. Only for one case (`case_2`), the property is not satisfied as shown in Figure 7. Indeed the paper is reviewed by Trudy twice. In one review, she rejects the paper while in another one she accepts it.

For every property, the LTL Checker partitions the set of cases into two sets: L^{OK} (the cases that satisfy the property) and L^{NOK} (the ones that do not). If $L^{NOK} = \emptyset$, the property holds. Otherwise, L^{NOK} provides counterexamples. Both sets can be saved and analyzed further. For example, it is possible to construct a process model or social network for L^{OK} or L^{NOK} . This may be helpful when analyzing (root) causes for violations of a desirable property.

6 Related Work

Although focusing on a completely different problem, the work reported in this paper is related to earlier work on process mining, i.e., discovering a process model based on some event log. The idea of applying process mining in the context of workflow management was first introduced in [10]. Cook and Wolf have investigated similar issues in the context of software engineering processes using different approaches [13]. Herbst and Karagiannis also address the issue of process mining in the context of workflow management using an inductive approach [22]. They use stochastic task graphs as an intermediate representation and generate a workflow model described in the ADONIS modeling language. Then there are several variants of the α algorithm [9,35]. In [9] it is shown that this algorithm can be proven to be correct for a large class of processes. In [35] a heuristic approach using rather simple metrics is used to construct so-called “dependency/frequency tables” and “dependency/frequency graphs”. This is used as input for the α algorithm. As a result it is possible to tackle the problem of noise. For more information on process mining we refer to a special issue of Computers in Industry on process mining [8] and a survey paper [7]. Given the scope of this paper, we are unable to provide a complete listing of the many papers on process mining published in recent years. Instead, we refer to our website www.processmining.org and the references there for a more elaborate overview.

Conformance testing, i.e., checking whether a given model and a given event log fit together, can be seen a very specific form of log-based verification. Instead of some logical formula, a process model (e.g., Petri net) is used to verify whether the log satisfies some behavioral properties. Therefore, the work of Cook et al. [14,12] is closely related to this paper. In [14] the concept of process validation is introduced. It assumes an event stream coming from the model and an event stream coming from real-life observations. Both streams are compared. Only in the last part of the paper an attempt is made to selectively try and match possible event streams of the model to a given event stream. As a result only fitness is considered and the time-complexity is problematic as the state-space of the model needs to be explored. In [12] the results are extended to include time aspects. The notion of conformance has also been discussed in the context of security [5], business alignment [2], and genetic mining [4].

Monitoring events with the goal to verify certain properties has been investigated in several domains, e.g., in the context of requirements engineering [16,31,32] and program monitoring [17,20,21]. It is also interesting to note the possible applications of such techniques in the context of monitoring web services. In such a distributed environment with multiple actors, it is highly relevant to be able to monitor the behavior of each actor.

The work of Robinson [31,32] on requirements engineering is highly related. He suggests the use of LTL for the verification of properties. Important differences between this approach and ours are the focus on real-time monitoring (with model-checking capabilities to warn for future problems) and the coding required to check the desired properties. The following quote taken from [31] illustrates the focus of this work: “Execution monitoring of requirements is a technique that tracks

the run-time behavior of a system and notes when it deviates from its design-time specification. Requirements monitoring is useful when it is too difficult (e.g., intractable) to prove system properties. To aid analysis, assumptions are made as part of the requirements definition activity. The requirements and assumptions are monitored at run-time. Should any such conditions fail, a procedure can be invoked (e.g., notification to the designer).” In a technical sense, the work of Havelund et al. [20,21] is highly related. Havelund et al. propose three ways to evaluate LTL formulas: (1) automata-based, (2) using rewriting (based on Maude), (3) and using dynamic programming. We use the latter approach (dynamic programming).

Process mining, conformance testing, and log-based verification can be seen in the broader context of Business (Process) Intelligence (BPI) and Business Activity Monitoring (BAM). In [18,19,34] a BPI toolset on top of HP’s Process Manager is described. The BPI tools set includes a so-called “BPI Process Mining Engine”. In [27] Zur Muehlen describes the PISA tool which can be used to extract performance metrics from workflow logs. Similar diagnostics are provided by the ARIS Process Performance Manager (PPM) [24]. The latter tool is commercially available and a customized version of PPM is the Staffware Process Monitor (SPM) which is tailored towards mining Staffware logs. Note that none of the latter tools is supporting conformance testing or the checking of (temporal) properties. Instead, the focus of these tools is often on performance measurements rather than monitoring (un)desirable behavior.

For a more elaborate description of the LTL language and checker we refer to manual of the LTL Checker [11]. Note that this is the first paper describing both. Moreover, it is the first paper discussing the application of log-based verification in the context of process mining.

7 Conclusion

This paper presents both a language and a tool to enable the verification of properties based on event logs. The language is based on LTL and is tailored towards events logs stored in the MXML format. The MXML format is a tool-independent format to log events and can be generated from audit trails, transaction logs and other data sets describing business events. Current software allows for the easy collection of such data, cf. BPM, WFM, CRM, BAM systems. Moreover, the need for both flexibility [1,15,29,30] and auditing capabilities (cf. the Sarbanes-Oxley Act [33]) underpins the relevance of the results presented in this paper.

We have predefined 60 typical properties one may want to verify (e.g., the 4-eyes principle). These can be used without any knowledge of the LTL language. In addition the user can define new sets of properties. These properties may be application specific and may refer to data. Each property is specified in terms of a so-called formula. Formulas may be parameterized, are reusable, and carry explanations in HTML format. This way both experts and novices may use the LTL Checker. The LTL Checker has been implemented in the ProM framework and the results can be further analyzed using a variety of process mining tools.

We do not propose to solely use LTL for the type of analysis discussed in this paper. In addition to the LTL Checker and the process mining tools, conventional tools such as SQL and XPath can also be used to query and filter event logs. For example, in the context of a case study we loaded MXML files into an Oracle database to query them using SQL. SQL allows for the analysis of simple questions like the 4-eyes principle but not for the easy formulation of temporal questions, e.g., `action_follows_decision` (cf. lines 16-18 in Table 2). Recently, we also developed a so-called *conformance checker* that measures the “difference” between a process model and a log (cf. www.processmining.org).

References

1. W.M.P. van der Aalst. Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change. *Information Systems Frontiers*, 3(3):297–317, 2001.
2. W.M.P. van der Aalst. Business Alignment: Using Process Mining as a Tool for Delta Analysis. In J. Grundspenkis and M. Kirikova, editors, *Proceedings of the 5th Workshop on Business Process Modeling, Development and Support (BPMDS'04)*, volume 2 of *Caise'04 Workshops*, pages 138–145. Riga Technical University, Latvia, 2004.
3. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
4. W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M. Weijters. Genetic Process Mining. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 48–69. Springer-Verlag, Berlin, 2005.
5. W.M.P. van der Aalst and A.K.A. de Medeiros. Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. In N. Busi, R. Gorrieri, and F. Martinelli, editors, *Second International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004)*, pages 69–84. STAR, Servizio Tipografico Area della Ricerca, CNR Pisa, Italy, 2004.
6. W.M.P. van der Aalst and M. Song. Mining Social Networks: Uncovering Interaction Patterns in Business Processes. In J. Desel, B. Pernici, and M. Weske, editors, *International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer-Verlag, Berlin, 2004.
7. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
8. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
9. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
10. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.

11. H. de Beer. *The LTL Checker Plugins: A Reference Manual*. Eindhoven University of Technology, Eindhoven, 2004.
12. J.E. Cook, C. He, and C. Ma. Measuring Behavioral Correspondence to a Timed Concurrent Model. In *Proceedings of the 2001 International Conference on Software Maintenance*, pages 332–341, 2001.
13. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
14. J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, 1999.
15. C.A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In N. Comstock, C. Ellis, R. Kling, J. Mylopoulos, and S. Kaplan, editors, *Proceedings of the Conference on Organizational Computing Systems*, pages 10 – 21, Milpitas, California, August 1995. ACM SIGOIS, ACM Press, New York.
16. S. Fickas, T. Beauchamp, and N.A.R. Mamy. Monitoring Requirements: A Case Study. In *Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE'02)*, page 299. IEEE Computer Society, 2002.
17. D. Giannakopoulou and K. Havelund. Automata-Based Verification of Temporal Properties on Running Programs. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'01)*, pages 412–416. IEEE Computer Society Press, Providence, 2001.
18. D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.C. Shan. Business process intelligence. *Computers in Industry*, 53(3):321–343, 2004.
19. D. Grigori, F. Casati, U. Dayal, and M.C. Shan. Improving Business Process Quality through Exception Understanding, Prediction, and Prevention. In P. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. Snodgrass, editors, *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 159–168. Morgan Kaufmann, 2001.
20. K. Havelund and G. Rosu. Monitoring Programs Using Rewriting. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'01)*, pages 135–143. IEEE Computer Society Press, Providence, 2001.
21. K. Havelund and G. Rosu. Synthesizing Monitors for Safety Properties. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *Lecture Notes in Computer Science*, pages 342–356. Springer-Verlag, Berlin, 2002.
22. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
23. T. Hoffman. Sarbanes-Oxley Sparks Forensics Apps Interest: Vendors Offer Monitoring Tools to Help Identify Incidents of Financial Fraud. *ComputerWorld*, 38:14–14, 2004.
24. IDS Scheer. ARIS Process Performance Manager (ARIS PPM): Measure, Analyze and Optimize Your Business Process Performance (whitepaper). IDS Scheer, Saarbruecken, Germany, <http://www.ids-scheer.com>, 2002.
25. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley, Reading MA, 1998.
26. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.

27. M. zur Mühlen and M. Rosemann. Workflow-based Process Monitoring and Controlling - Technical and Organizational Issues. In R. Sprague, editor, *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS-33)*, pages 1–10. IEEE Computer Society Press, Los Alamitos, California, 2000.
28. A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th IEEE Annual Symposium on the Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, Providence, 1977.
29. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
30. S. Rinderle, M. Reichert, and P. Dadam. Correctness Criteria For Dynamic Changes in Workflow Systems: A Survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.
31. W.N. Robinson. Monitoring Software Requirements using Instrumented Code. In *Proceedings of the 35th Annual Hawaii IEEE International Conference on Systems Sciences*, pages 276–276. IEEE Computer Society , 2002.
32. W.N. Robinson. Monitoring Web Service Requirements. In *Proceedings of 11th IEEE International Conference on Requirements Engineering (RE 2003)*, pages 56–74. IEEE Computer Society , 2003.
33. P. Sarbanes, G. Oxley, and et al. Sarbanes-Oxley Act of 2002, 2002.
34. M. Sayal, F. Casati, U. Dayal, and M.C. Shan. Business Process Cockpit. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. Morgan Kaufmann, 2002.
35. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.

A Detailed Investigation of Memory Requirements for Publish/Subscribe Filtering Algorithms

Sven Bittner and Annika Hinze

University of Waikato, New Zealand

{s.bittner, a.hinze}@cs.waikato.ac.nz

Abstract. Various filtering algorithms for publish/subscribe systems have been proposed. One distinguishing characteristic is their internal representation of Boolean subscriptions: They either require conversions into DNFs (canonical approaches) or are directly exploited in event filtering (non-canonical approaches).

In this paper, we present a detailed analysis and comparison of the memory requirements of canonical and non-canonical filtering algorithms. This includes a theoretical analysis of space usages as well as a verification of our theoretical results by an evaluation of a practical implementation. This practical analysis also considers time (filter) efficiency, which is the other important quality measure of filtering algorithms. By correlating the results of space and time efficiency, we conclude when to use non-canonical and canonical approaches.

1 Introduction

Publish/subscribe (pub/sub) is a communication pattern targeting on the active notification of clients: Subscribers define Boolean subscriptions to specify their interests; publishers disseminate their information by the help of event messages containing attribute/value pairs. A pub/sub system is acting as broker; it filters all incoming event messages and notifies subscribers if their registered subscriptions are matching. An integral and essential part of pub/sub systems is this filtering process, i.e., the determination of all subscribers interested in an incoming event message (also referred to as primitive event filtering). Generally, filtering algorithms for pub/sub systems should fulfil two requirements [6]:

- Efficient event filtering (fast determination of interested subscribers)
- Scalable event filtering (supporting large numbers of subscriptions)

For efficiency, current pub/sub systems apply main memory filtering algorithms. Thus, we can directly deduce the scalability characteristics of the central components of these systems from their memory requirements [2, 3, 10]. This characteristic implies the need to economize the usage of memory resources.

We can distinguish between two classes of filtering approaches for pub/sub systems: (i) algorithms directly filtering on Boolean subscriptions [3, 4, 11] (referred to as non-canonical approaches in the following), and (ii) algorithms filtering on subscriptions in canonical forms [1, 5, 6, 8, 12] (referred to as canonical approaches). Internally,

algorithms of Class (ii) either filter on disjunctive normal forms (DNF) [5] or only support conjunctive subscriptions [1, 6, 8, 12]. Thus, if supporting arbitrary Boolean subscriptions, these approaches always require conversions of subscriptions to DNFs. If algorithms only allow conjunctions, each disjunctive element of a DNF is treated as a separate subscription [9].

Canonical approaches store subscriptions plainly as canonical forms. Hence, if subscriptions merely utilize such forms, this class of algorithms allows for efficient event filtering. This results from the ability to neglect arbitrary Boolean expressions while filtering. However, due to the need of converting Boolean subscriptions to DNFs, subscriptions consume more space than required by their original forms [3]. Additionally, the matching process works over more (or, in case of supporting DNFs, larger) subscriptions. For non-canonical approaches holds the opposite: Subscriptions demand less memory for storage but involve a more sophisticated matching. Hence, the benefits and drawbacks (you can find a more detailed discussion in [3]) of both classes of filtering algorithms are twofold and necessitate a thorough analysis to allow solid statements about their advantages and disadvantages.

In this paper, we present a thorough analysis and evaluation of the memory requirements of canonical and non-canonical filtering algorithms. This includes a theoretical analysis as well as a practical investigation of space usages. Furthermore, we correlate the memory requirements of the analyzed algorithms to their filter efficiency (time efficiency). As representatives of canonical algorithms we analyzed the counting [1, 12] and the cluster approach [6, 8], which are known to be efficient and reasonably memory-friendly [3]. Non-canonical algorithms are represented by the filtering approach in [3] because of its time efficiency due to the utilization of indexes. Our decision to compare these particular algorithms is also driven by their similar exploitation of one-dimensional indexes for filtering. In detail our contributions in this paper are:

1. A characterization scheme for qualifying primitive subscriptions
2. A theoretical analysis and comparison of the memory requirements of canonical and non-canonical filtering algorithms
3. A practical verification of our theoretical results of memory usages
4. A correlation of memory usage and filter efficiency of filtering algorithms
5. Recommendations for the utilization of non-canonical and canonical algorithms

The rest of this paper is structured as follows: Section 2 gives an overview of the analyzed algorithms and presents related work. Our characterization scheme qualifying subscriptions can be found in Sect. 3 as well as our theoretical analysis of memory requirements. Section 4 includes a comparison of the theoretical memory usages and their graphical presentation. We practically verify our results in Sect. 5.1, followed by the correlation of memory usage to filtering efficiency in Sect. 5.2. Finally, we conclude and present our future work in Sect. 6.

2 Analyzed Algorithms and Related Work

In this section, we outline the three filtering algorithms used in our later analysis. Afterwards, we present related work that is evaluating and comparing different filtering approaches in Sect. 2.2.

2.1 Review of Analyzed Algorithms

We now give a brief overview of the algorithms analyzed in Sect. 3, namely the counting algorithm [1, 12], the cluster algorithm [6, 8], and the non-canonical algorithm [3]. We have chosen these algorithms for our analysis due to their time and space efficiency characteristics¹. We restrict this subsection to a short review of the approaches and refer to the original works for thorough study and description of the algorithms.

Review of the Counting Algorithm. The counting algorithm was originally proposed in [12] for filtering on plain text in combination with secondary storage. Later, it was adopted as pure main memory filtering approach working on attribute/value pairs, e.g., [1]. It only supports conjunctive subscriptions and requires the conversion of subscriptions involving disjunctions into DNFs. Then, each element (i.e., a conjunction) participating in the one disjunction of a DNF is treated as separate subscription [9].

Filtering works in two steps: Firstly, matching predicates are determined by utilizing one-dimensional indexes (predicate matching). Secondly (subscription matching), all subscriptions involving these predicates are derived by exploiting a predicate subscription association table. Counters in hit vectors are increased for each matching predicate per subscription. Finally, hit and subscription predicate count vector are compared.

Review of the Cluster Algorithm. The cluster algorithm is described in detail in [6] and is based on the algorithm presented in [8]. Similar to the counting algorithm, only conjunctive subscriptions are supported by this approach. This requires a conversion to DNFs when supporting arbitrary Boolean subscriptions. Subscriptions are grouped into clusters according to their access predicates² and total number of predicates.

Again, event filtering works in two steps: In predicate matching all matching predicates are determined by the help of one-dimensional indexes. For all matching access predicates, clusters with potentially matching subscriptions can be found by utilizing a cluster vector. Then, the subscriptions inside these clusters are evaluated by testing if all their predicates have been fulfilled (subscription matching).

Review of the Non-Canonical Algorithm. The non-canonical algorithm is presented in [3]. It comprises no restriction to conjunctive subscriptions as the previous two approaches. Instead, it directly exploits the Boolean expressions used in subscriptions.

Also the non-canonical algorithm utilizes one-dimensional indexes to efficiently determine predicates matching incoming events. Subscriptions are encoded in subscription trees representing their Boolean structure and involved predicates. A minimum predicate count vector states the minimal number of fulfilled predicates required for each subscription to match (variation of [3]). A hit vector is used to accumulate the number of fulfilled predicates per subscription (also a variation from [3]).

Once more, this matching approach involves a two-step event filtering process beginning with the determination of matching predicates (predicate matching). Then, by the help of a predicate subscription association table all potential candidate subscriptions are determined (subscription matching step). The work in [3] proposes to evaluate

¹ Refer to [2] and Sect. 1 for a more detailed argumentation.

² Common predicates that have to be fulfilled by an event to lead to fulfilled subscriptions.

all candidate subscriptions. However, if using a minimum predicate count vector, only subscriptions with more than the minimum number of matching predicates (minimum number of fulfilled predicates required for matching $|p_{min}|$) have to be evaluated. An example is the following: If a subscription consists of three disjunctive elements that contain conjunctions of nine, five, and seven predicates it holds $|p_{min}| = 5$. The accumulation of matching predicates per subscription is obtained using a hit vector.

2.2 Previous Evaluations

There have already been some comparative evaluations of event filtering approaches for primitive events. However, nearly all of them merely target evaluations of time efficiency in specifically chosen settings. Additionally, a detailed theoretical analysis of memory requirements of filtering algorithms cannot be found so far. The results of current practical evaluations of space efficiency are too restricted to be generalizable.

In [1] several implementations of the counting algorithm have been evaluated, but there is no comparison of this approach to other filtering solutions. For the investigation of subscription matching, subscriptions consist of only one to five predicates over domains of only ten different values. Thus, we cannot generalize the results of [1] to more complex and sophisticated settings utilizing expressive Boolean subscription languages. Additionally, a satisfactory theoretical evaluation is missing.

The work in [6] compares implementations of counting and cluster algorithm. However, the assumptions are similarly restricted as in [1]: only five predicates are used per subscription, domains consist of thirty-five possible values. Furthermore, subscriptions mainly define equality predicates in [6]. Naturally, this leads to a well-performing cluster algorithm, which is specifically designed to exploit this characteristic. Hence, the results of [6] do not present general settings and are mainly targeting filter efficiency.

In [3] the counting and the non-canonical approach are compared briefly. Thus, this analysis allows only limited conclusions about the behaviors of these algorithms. Again, a theoretical analysis of memory requirements is missing.

3 Theoretical Characterization and Analysis of Memory Usage

In this section we firstly present our characterization scheme allowing for a general representation of Boolean subscriptions. Our theoretical analysis of memory requirements based on our characterization can then be found in Sect. 3.2.

3.1 Characterization of Boolean Subscriptions

We now present our approach of characterizing subscriptions (and their management in algorithms). Since we target an evaluation of memory requirements, our methodology is based on attributes affecting the memory usage for storing subscriptions for efficient event filtering. Our approach also allows for a successful representation of the space requirements of the three filtering algorithms presented in Sect. 2.1.

We have identified 14 parameters, which are compactly shown in Table 1. The parameters of Class S allow for both a representation of the characteristics of subscriptions and a determination of their memory requirements for index structures. These six parameters directly describe subscriptions in their quantity $|s|$ and their average number

Table 1. Overview of parameters characterizing subscriptions (Class S – subscription-related, Class A – algorithm-related, Class C – conversion-related, Class E – subscription-event-related)

Symbol	Parameter Name (Calculation)	Class
$ p $	Number of predicates per subscription	S
$ op $	Number of Boolean operators per subscription	S
op^r	Relative number of Boolean operators per subscription ($op^r = \frac{ op }{ p }$)	S
$ s $	Number of subscriptions	S
$ p_u $	Number of unique predicates	S
r_p	Predicate redundancy ($r_p = 1.0 - \frac{ p_u }{ p s }$)	S
$w(s)$	Width of subscription identifiers	A
$w(p)$	Width of predicate identifiers	A
$w(l)$	Width of subscription locations	A
$w(c)$	Width of cluster references	A
S_s	Number of disjunctively combined elements after conversion	C
s_p	Number of conjunctive elements per predicate after conversion	C
s^r	Relative no. of conjunctive elements per predicate after conversion ($s^r = \frac{s_p}{S_s}$)	C
p_e	Number of fulfilled predicates per event	E

of predicates $|p|$ and operators $|op|$. Parameter op^r expresses the number of operators relatively to the number of predicates. To determine predicate redundancy r_p , we also require the number of unique predicates registered with the system $|p_u|$.

Class A of parameters explicitly deals with filtering algorithm-related characteristics influencing the internal storage of subscriptions. The behavior of canonical approaches is expressed by the three parameters of Class C. The number of disjunctively combined elements in a converted DNF (which have to be treated as separate subscriptions [9]) is described by S_s . The average number of such elements containing a predicate from an original subscription s_p also strongly influences the behavior of canonical approaches. We can combine these two parameters to the relative number of conjunctive elements per predicate s^r . The parameter p_e of Class E incorporates the relation between subscriptions and events, which influences both space and time efficiency of filtering algorithms. For a detailed description of these parameters, we refer to [2].

Altogether, these fourteen parameters allow to characterize subscriptions, to derive the major memory requirements of filtering algorithms, and to describe the relation between events and subscriptions affecting the time efficiency of event filtering.

3.2 Theoretical Analysis of Memory Requirements

After presenting the parameters required to analyze the memory usage of filtering algorithms, we now continue with our theoretical analyzes. Note that our theoretical obser-

vations do not take into account implementation issues and other practical considerations. Our results are a base line helping to find a suitable filtering algorithm. An actual comparison of the theoretical memory requirements can be found in Sect. 4 as well as considerations for practical implementations.

Theoretical Memory Analysis of the Counting Algorithm. We now analyze the memory requirements of the counting algorithm [1, 12] in respect to the characterizing parameters defined in Sect. 3.1. According to [1] and our review in Sect. 2.1, the counting algorithm requires a fulfilled predicate vector, a hit vector, a subscription predicate count vector, and a predicate subscription association table. To efficiently support unsubscriptions, we also necessitate a subscription predicate association table. In the following we describe these data structures and derive their minimal memory requirements. We start our observations for cases with no predicate redundancy ($r_p = 0.0$). Subsequently, we extend our analysis to more general settings involving predicate redundancy.

Fulfilled predicate vector: The fulfilled predicate vector is required to store matching predicates in the predicate matching step. In an implementation, we might apply an ordinary vector ($p_e w(p)$ bytes) or a bit vector implementation ($\frac{|p||s|}{8}$ bytes) depending on the proportion of matching predicates.

In cases of high predicate redundancy there is only a small number of unique predicates. Thus, a bit vector implementation might require less memory compared to an ordinary vector implementation. However, if the fraction of fulfilled predicates per event p_e and totally registered predicates ($|p||s|$ predicates in total) is quite small, utilizing an ordinary vector might be advantageous.

Hit vector: The hit vector accumulates the number of fulfilled predicates per subscription. For simplicity, we assume a maximum number of 255 predicates per subscription (we can easily relax this assumption). Thus, each entry in the hit vector requires 1 byte. Altogether, for $|s|$ subscriptions creating S_s disjunctively combined elements due the canonical conversion, the space requirements are $|s|S_s$ bytes for the hit vector. Since this vector consists of one entry per subscription, its memory usage is independent of predicate redundancy r_p .

Subscription predicate count vector: We also require to store the total number of predicates each subscription consists of. According to our assumption for the hit vector, each subscription can be represented by a 1-byte entry. Thus, we require $|s|S_s$ bytes in total due to the applied canonical conversions (cp. hit vector).

Similar to the hit vector, the subscription predicate count vector does not depend on predicate redundancy r_p (it consists of entries per subscription).

Predicate subscription association table: This table has to be applied to efficiently find all subscriptions a predicate belongs to. In an implementation, each predicate has to be mapped to a list of subscriptions due to the required canonical conversions. This also holds in cases of no predicate redundancy ($r_p = 0$). Least memory is demanded if predicate identifiers might be used as indexes in this table (this requires consecutive predicate identifiers). For storing the list of subscriptions we have to store the corresponding number of subscription identifiers at a minimum (neglecting additional implementation overhead, such as the length of each list). Thus, altogether we have to record the list of subscription identifiers (requiring

$w(s)s_p$ bytes per predicate) for all registered predicates ($|p||s|$ predicates in total), which requires $w(s)s_p|p||s|$ bytes in total.

If considering predicate redundancy r_p , for unique predicates (including one of each redundant predicate) the following amount of memory is required in bytes: $(1.0 - r_p)w(s)s_p|p||s|$. Redundant predicates use $r_p w(s)s_p|p||s|$ bytes. Thus, r_p does not influence the size of the predicate subscription association table.

Subscription predicate association table: The previously described data structures are required to support an efficient event filtering. However, unsubscription are supported very inefficiently. This is due to missing associations between subscriptions and predicates [1].

Least memory for subscription predicate associations is utilized when using a subscription identifier as index in a subscription predicate association table. Each entry maps this identifier to a list of predicate identifiers (there is also some implementation overhead as described for the previous table). Thus, we have to store a list of predicates for each subscription ($|s|S_s$ subscriptions in total due to conversions). Each list has to hold $|p|\frac{s_p}{S_s}$ predicate identifiers, which leads to $w(p)|s|S_s|p|\frac{s_p}{S_s} = w(p)|s||p|s_p$ bytes in total for a subscription predicate association table.

Predicate redundancy r_p does not influence this table because it contains entries for each subscription. Thus, redundant predicates do not allow for the storage of less associations between subscriptions and predicates.

When accumulating the former memory usages, we require the following amount of memory in bytes (we exclude the fulfilled predicate vector since it is utilized by all three analyzed algorithms)

$$\text{mem}_{\text{counting}} = |s|(2S_s + w(s)s_p|p| + w(p)s_p|p|) . \quad (1)$$

This observation holds in all cases of predicate redundancy r_p as shown above.

Theoretical Memory Analysis of the Cluster Algorithm. This section presents an evaluation of the memory requirements of the cluster algorithm [6, 8] according to the characterizing parameters defined in Sect. 3.1. However, this algorithm has several restrictions (e.g., usage of highly redundant equality predicates) and strongly depends on the subscriptions actually registered with the pub/sub system. Thus, we are not able to express all memory requirements of this algorithm based on our characterization scheme. In our following analysis we neglect the space usage of some data structures (cluster vector, references to cluster vector) and focus on the most space consuming ones, which leads to an increased amount of required memory in practice.

To efficiently support unsubscriptions, [6] suggests to utilize a subscription cluster table to determine the cluster each subscription is stored in. In our opinion, this data structure is not sufficient for a fast removal of subscriptions: The subscription cluster table allows for the fast determination of the cluster a subscription is stored in. Thus, we are able to remove subscriptions from clusters. It remains to determine when predicates might be removed from index structures due to the inherent assumption of predicate redundancy in [6]³. Also the necessity of canonical conversions leads to shared pred-

³ The motivation for [6] is the existence of shared predicates (predicate redundancy) because the clustering of subscriptions is obtained via access predicates, i.e., predicates need to be shared.

icates. Thus, to allow for a deletion of predicates in index structures, we require an association between predicates and subscriptions utilizing these predicates, e.g., by the application of a predicate subscription association table or by storing these associations inside index structures themselves.

The memory requirements of the cluster algorithm are as follows. Again, we firstly derive the space usage of the algorithm in case of no predicate redundancy ($r_p = 0.0$). Secondly, we generalize our results to cases involving predicate redundancy.

Predicate bit vector: This vector is similar to the fulfilled predicate vector applied in the counting algorithm. However, we require a bit vector implementation (as stated in [6]) due to the requirement of accessing the state of predicates (fulfilled or not fulfilled) directly. Thus, we demand $\frac{|p||s|}{8}$ bytes for the predicate bit vector. High predicate redundancy does not influence these memory requirements.

Clusters: Subscriptions themselves are stored in clusters according to both their access predicates and their total number of predicates. Clusters consist of a subscription line storing an identifier for each subscription ($w(s)$ bytes required per subscription). Furthermore, they contain a predicate array holding the predicates each subscription consists of (on average $\frac{s_p}{S_s}|p|w(p)$ bytes per subscription if only storing predicate identifiers). Clusters storing subscriptions with the same number of predicates and access predicates are linked together in a list structure. However, we neglect the memory requirements for this implementation-specific attribute.

Altogether, clusters require $|s|S_s(w(s) + \frac{s_p}{S_s}|p|w(p))$ bytes to store $|s|S_s$ subscriptions. Predicate redundancy does not influence the size of clusters. This results from the observation that clusters store predicates for all subscriptions. This storage happens in all cases of r_p and does not vary according to the uniqueness of predicates.

Subscription cluster table: This table is an additional data structure required to support efficient unsubscriptions (see argumentation above). It allows for the determination of the cluster each subscription is stored in. Utilizing subscription identifiers as indexes for the subscription cluster table, we require $|s|S_sw(c)$ bytes for the storage of $|s|S_s$ cluster references. Also this table is focussed on a mapping of subscriptions. Thus, its size is independent of predicate redundancy r_p .

Predicate subscription association table: As shown above, an association between predicates and subscriptions is required to allow for an efficient support of unsubscriptions. This information could be stored in a separate predicate subscription association table or as part of indexes themselves. Both options require the same amount of additional memory. If using predicate identifiers as indexes (or storing associations inside indexes), we require $w(s)s_p|p||s|$ bytes for these associations of $|p||s|$ predicates. Each predicate is contained in $w(s)s_p$ subscriptions on average. Similar to our observation for the counting algorithm, predicate redundancy does not influence the size of predicate subscription associations.

Accumulating the memory requirements of the formerly mentioned data structures (excluding the predicate bit vector) leads to the following number of bytes

$$\text{mem}_{\text{cluster}} = |s|(S_sw(s) + s_p|p|w(p) + S_sw(c) + s_p|p|w(s)) . \quad (2)$$

Again, our observation represents the memory requirements of the cluster algorithm regardless of predicate redundancy r_p as shown before.

Theoretical Memory Analysis of the Non-canonical Algorithm. As last algorithm for our analysis, we have chosen a variant of the non-canonical approach [3] as presented in Sect. 2. According to [3], inner nodes of subscription trees store Boolean operators and leaf nodes store predicate identifiers. Each leaf node requires $w(p)$ bytes to store its predicate identifier and 1 byte to denote itself as a leaf node. For inner nodes, we store the Boolean operator in 1 byte and use 1 byte to denote the number of children (this implies that at least 255 predicates are supported per subscription as in the other algorithms presented before). In contrast to [3], we do not store the width of the children of inner nodes in bytes. Hence, to access the last out of n children, we have to compute the widths of all $n - 1$ previously stored children.

The non-canonical approach inherently supports efficient unsubscriptions due to its characteristic to store associations between subscriptions and predicates and vice versa. In the following, we analyze the memory requirements of the non-canonical approach beginning with the case of no predicate redundancy ($r_p = 0.0$). Afterwards, our analysis is extended to general settings involving predicate redundancy $r_p > 0$.

Fulfilled predicate vector: This vector serves the same purpose as its counterpart in the counting algorithm. Therefore, it requires the same amount of memory according to its realization and depending on $r_p \min(\frac{|p||s|}{8}, p_e w(p))$ bytes.

Subscription trees: The encoding of subscription trees has been presented above. For predicates stored in leaf nodes, we require $(w(p) + 1)|p|$ bytes per subscription. Inner nodes demand $2|op|$ bytes of memory for each subscription. Thus, for all registered subscriptions, we need $|s|((w(p) + 1)|p| + 2|op|)$ bytes. Subscription trees have to store operators and predicate identifiers in all cases. Thus, they do not depend on predicate redundancy r_p .

Subscription location table: This table is applied to associate subscription identifiers and subscription trees. If utilizing subscription identifiers as indexes in this table (consecutive identifiers necessitated), we require $w(l)|s|$ bytes. Since the subscription location table stores entries per subscription, its memory usage is not influenced by predicate redundancy r_p .

Predicate subscription association table: The predicate subscription association table requires less memory than its counterparts in the previously analyzed algorithms. This is implied by the fact that subscriptions do not need a conversion in canonical forms. Thus, predicates are involved in less subscriptions (only one subscription in case of $r_p = 0.0$). Altogether, we require $|s||p|w(s)$ bytes for the predicate subscription association table.

Similar to the counterparts of this table in the two other algorithms, the memory usage of the predicate subscription association table is independent of predicate redundancy r_p .

Hit vector: Similar to the hit vector in the counting approach, this vector accumulates the number of fulfilled predicates per subscription. The hit vector requires $|s|$ bytes of memory since no conversions to canonical expressions are required by the non-canonical approach and according to the common assumption of a maximum of 255 predicates per subscription.

Minimum predicate count vector: This vector stores the minimum number of predicates per subscription that are required to be fulfilled $|p_{min}|$ in order to lead to

a fulfilled subscription. According to our assumption of a maximum of 255 predicates per subscription, the minimum predicate count vector requires $|s|$ bytes of memory.

The required data structures (excluding the fulfilled predicate vector) sum up to the following amount of memory in bytes

$$\text{mem}_{\text{non-canonical}} = |s|(w(p)|p| + |p| + 2|op| + w(l) + |p|w(s) + 2) . \quad (3)$$

Analogous to the previously described algorithms, the theoretical memory usage of the non-canonical approach does not depend on r_p as shown in our analysis.

4 Comparison of Theoretical Memory Requirements

After our analysis of three filtering algorithms and the derivation of their theoretical memory requirements, we now compare the memory usage of the two canonical approaches (counting and cluster algorithm) to the non-canonical algorithm. From this analysis we can deduce under which circumstances a non-canonical approach should be preferred (in respect to memory usage and thus scalability) and which settings favor canonical filtering algorithms.

In our following analysis, we focus on differing data structures of algorithms, i.e., we neglect the fulfilled predicate/predicate bit vector, which is incidentally required by all three algorithms. Thus, we directly compare (1) to (3).

All memory requirements derived in the last section grow linearly with increasing numbers of subscriptions. Moreover, all of them cut the ordinate in zero. Hence, for a comparison we solely need to analyze the first derivations of (1) to (3) in $|s|$

$$\text{mem}'_{\text{counting}}(|s|) = 2S_s + w(s)s_p|p| + w(p)s_p|p| . \quad (4)$$

$$\text{mem}'_{\text{cluster}}(|s|) = S_s w(s) + s_p|p|w(p) + S_s w(c) + s_p|p|w(s) . \quad (5)$$

$$\text{mem}'_{\text{non-canonical}}(|s|) = w(p)|p| + |p| + 2|op| + w(l) + |p|w(s) + 2 . \quad (6)$$

To eliminate some parameters, let us assume fixed values for parameters of Class A: $w(s) = 4$, $w(p) = 4$, $w(l) = 4$, and $w(c) = 4$, i.e., the widths of subscription identifiers, predicate identifiers, subscription locations and cluster references are 4 bytes each. Furthermore, let us reduce the number of characterizing parameters specifying fixed values by utilizing the relative notions of op^r and s^r as introduced in Sect. 3.1.

We now compare the memory requirements (using the gradients) of the canonical algorithms (Equations (4) and (5)) to the memory requirements of the non-canonical approach (Equation (6)). We use the following notation to denote the canonical algorithm compared to the non-canonical approach: $S_s(\frac{\text{algorithm}}{\text{non-canonical}})$.

The inequalities shown in the following denote the point when the non-canonical approach requires less memory for its event filtering data structures than the respective canonical solution. These points are described in terms of the characterizing parameter S_s , i.e., if more than the stated number of disjunctively combined elements is created by the canonical conversion to DNF, the non-canonical approach requires less memory.

$$S_s(\frac{\text{counting}}{\text{non-canonical}}) > \frac{|p|(2op^r + 9) + 6}{2 + 8s^r|p|} . \quad (7)$$

$$S_s \left(\frac{\text{cluster}}{\text{non-canonical}} \right) > \frac{|p|(2op^r + 9) + 6}{8 + 8s^r|p|} . \quad (8)$$

In the following subsection we illustrate these observations graphically.

4.1 Graphical Illustration of Interchanging Memory Requirements

After the determination if the inequalities denoting the point when a non-canonical approach requires less memory than canonical algorithms (Equations (7) and (8)), we now present this turning point graphically.

Figure 1 shows the point of interchanging memory requirements for the counting algorithm and the cluster algorithm. We have chosen $op^r = 1.0$ in Fig 1(a) and Fig. 1(b); the parameter s^r is varied from 0.3 to 0.7. The abscissae of both figures show the number of predicates per subscription $|p|$, the ordinates are labeled with the number of disjunctively combined elements per subscription after conversion S_s . Both graphs denote which number of disjunctively combined elements have to be created by canonical conversions to DNF to favor the non-canonical approach in respect to memory requirements (cf. (7), (8)).

We can realize that the counting algorithm requires less memory in cases of small predicate numbers $|p|$ than the cluster algorithm. However, with increasing predicate numbers $|p|$ both algorithms behave nearly the same, i.e., for 50 or more predicates per subscription, it holds $S_s \approx 2.0$ (counting) and $S_s < 2.0$ (cluster) in case of $s^r = 0.7$. Thus, even if DNFs only consist of approximately 2 disjunctively combined elements, a non-canonical approach requires less memory. Smaller values of s^r favor the counting and the cluster algorithm. This is due to the fact of requiring less associations between predicates and subscriptions in these cases.

In Fig. 1(a) and Fig. 1(b), we have chosen $op^r = 1.0$, which describes the worst case scenario of the non-canonical algorithm. In practice, it always holds $op^r < 1.0$, since each inner node of a subscription tree has at least two children. Hence, a subscription tree containing $|p|$ leaf nodes (i.e., predicates) consists of a maximum of $|p| - 1$ inner nodes (i.e., operators). This implies $op^r \leq \frac{|p|-1}{|p|} < 1.0$. In practice, we have to expect much smaller values than 1.0 for op^r , because in subscription trees consecutive binary operators can be subsumed to n-ary ones.

These observations for the characterizing parameter op^r lead to further improved memory characteristics of the non-canonical approach. Figure 1(c) shows this behavior using $op^r = 0.5$ for the counting approach; the cluster algorithms is presented in Fig. 1(d). Thus, even if subscriptions use only one disjunction, a non-canonical approach shows less memory usage and better scalability than the counting algorithm ($s^r = 0.7$).

4.2 Considerations in Practice

Our previous analysis shows a comparison of the theoretical memory requirements of three algorithms. However, a practical implementation requires additional space for managing data structures, e.g., to link lists together, store lengths of variable-sized arrays, or practically realize hash tables. Thus, a practical implementation implies increasing space requirements of filtering algorithms compared to our theoretical analysis.

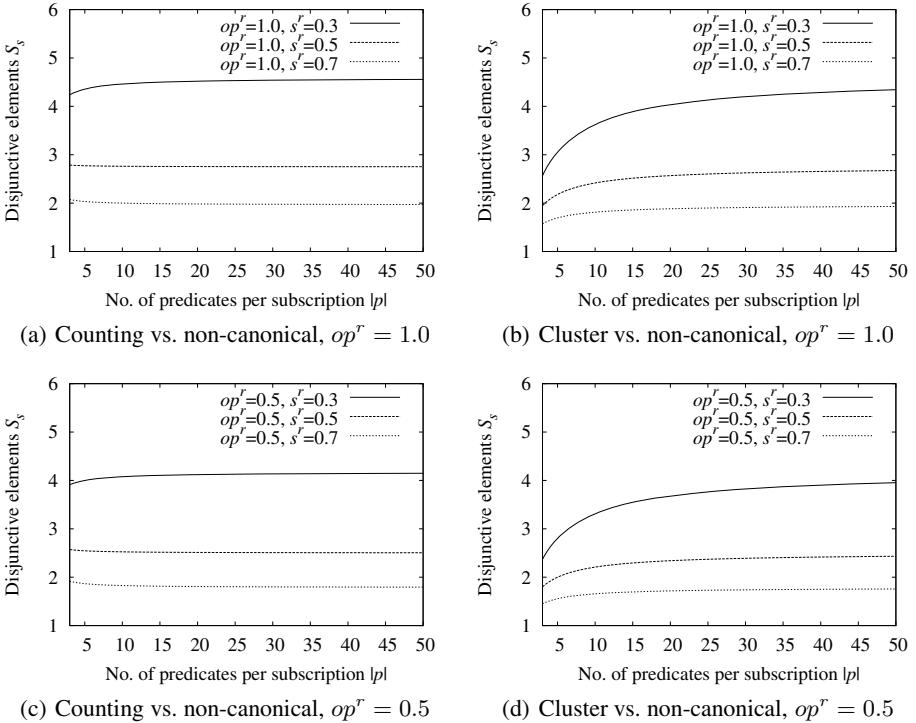


Fig. 1. Theoretically required number of disjunctively combined elements S_s to achieve less memory usage in the non-canonical approach compared to the counting and cluster algorithm using $op^r = 1.0$ (Fig. 1(a) and Fig. 1(b)) and $op^r = 0.5$ (Fig. 1(c) and Fig. 1(d))

Next to this general increase in memory requirements, data structures have to be implemented in a reasonable way, e.g., they have to support dynamic growing and shrinking if this is demanded in practice. Generally, constantly required data structures require a dynamic implementation. For data structures solely used in the event filtering process, i.e., fulfilled predicate, predicate bit and hit vector, a static implementation is sufficient due to the requirement of initializing them for each filtered event.

We have implemented such dynamic data structures in a space-efficient manner for our practical evaluation. Their comparison to the memory requirements of standard implementations, i.e., STL hash (multi) sets, has resulted in much less space consumptions for our implementations. We present our practical analysis in the next section.

5 Practical Analysis of Memory Requirements and Efficiency

In Sect. 3 and 4 we have presented a theoretical investigation of memory requirements of filtering algorithms and described the influence of a practical implementation on our theoretical results. In this section we extend our theoretical work and show the applicability of our theoretical results to practical settings (Sect. 5.1). Furthermore, we

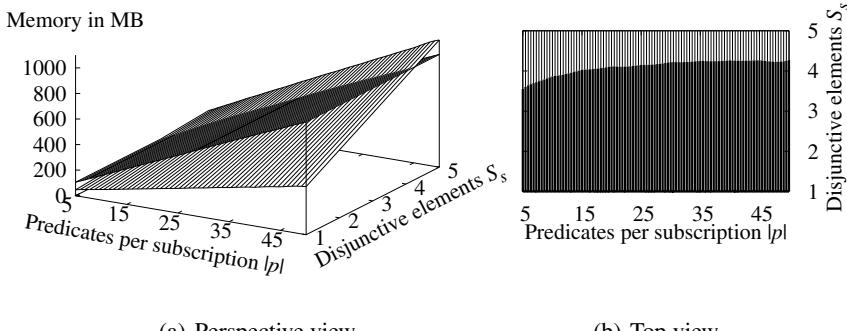


Fig. 2. Memory requirements in our practical experiments in case of $s^r = 0.3$, $op^r = 0.5$ and $|s| = 1,000,000$ (cf. Fig. 1(c) for theoretical results in the same scenario)

present a brief comparison of efficiency characteristics of the compared algorithms (Sect. 5.2).

We compare the non-canonical approach to one canonical algorithm by experiment. Because of the restrictions of the cluster approach (cf. Sect. 3.2), we have chosen the counting algorithm as representative of canonical algorithms for our practical analysis. This allows the generalization of our results to other settings than equality predicate-based application areas and areas dealing with less predicate redundancy as assumed by the cluster approach. Furthermore, the counting algorithm behaves more space efficient than the cluster approach (cf. Fig. 1). In a practical implementation the cluster approach without efficiently supported unsubscriptions and the counting approach show nearly the same memory requirements [6].

5.1 Practical Analysis of Memory Requirements

In this section, we compare the memory requirements of the counting algorithm and the non-canonical approach. Our actual implementations of these algorithms follow our descriptions in Sect. 4.2.

In our experiments, we want to verify our results shown in Fig. 1(c), i.e., in case of $op^r = 0.5$. Here we present the memory usage of the required data structures⁴ in case of 1,000,000 registered subscriptions with a growing number of predicates per subscription $|p|$ and a growing number of disjunctively combined elements after conversion S_s . For the parameter s^r (relative number of conjunctive elements per predicate after conversion), we have chosen to present the cases $s^r = 0.3$ and $s^r = 0.7$.

Our results are presented in three-dimensional figures. Figure 2(a) shows both algorithms in case of $s^r = 0.3$; Fig. 3(a) presents the case of $s^r = 0.7$. The x-axes in the figures represent the number of predicates per subscription $|p|$ ranging from 5 to 50, z-axes show the number of disjunctively combined elements after conversion S_s in the

⁴ We show the total memory requirements of our filtering process to allow for the incorporation of all influencing parameters, e.g., heap management structures.

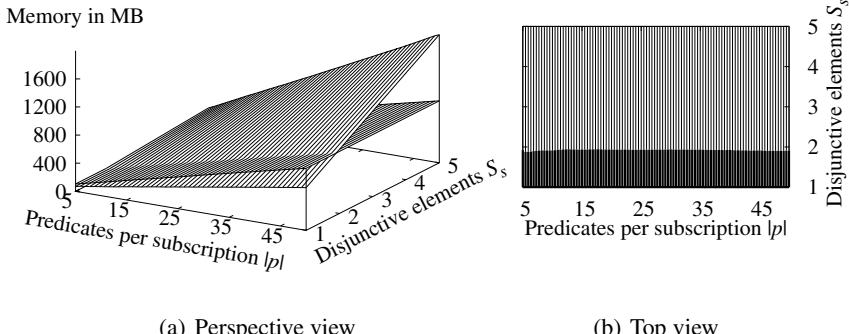


Fig. 3. Memory requirements in our practical experiments in case of $s^r = 0.7$, $op^r = 0.5$ and $|s| = 1,000,000$ (cf. Fig. 1(c) for theoretical results in the same scenario)

range of 1 to 5. The actually required amount of memory for holding the required data structures is illustrated at the y-axes of the figures.

There are two surfaces shown in each of the figures. The brighter ones illustrate the behaviors of the counting algorithm, the darker ones represent the non-canonical approach. As shown in our theoretical analysis, the non-canonical approach does not change its memory usage with growing S_s . Thus, its surface does always show the same memory requirements (y-axis) regardless of S_s (z-axis), e.g., approx. 900 MB for $|p| = 50$. This holds for both figures, Fig. 2(a) and 3(a), since the memory requirements of the non-canonical approach are independent of s^r . The counting algorithm, however, shows increasing memory requirements with growing S_s as described in (1). Furthermore, according to (1), increasing s_p (and thus s^r) results in advanced space usage.

As depicted in our theoretical comparison in Fig. 1(c), there exists a point of interchanging memory requirements of canonical and non-canonical algorithms. This point is denoted by a cutting of the surfaces of the two algorithms. In Fig. 2(a), this cutting occurs at $S_s \approx 4$, in Fig. 3(a) it happens at $S_s \approx 2$. To exactly determine the point of cutting surfaces we present a top view of the diagrams in Fig. 2(b) and Fig. 3(b), respectively. Figure 2(b) shows that the point of interchanging memory requirements can be found between $S_s = 3$ and $S_s = 5$ dependent on $|p|$. For $s^r = 0.7$ (Fig. 3(b)), it is always located slightly below $S_s = 2$. Comparing these practical results to our theoretical results in Fig. 1(c), we realize that our theoretical analysis has predicted nearly the same behavior of the two algorithms: Even if only 2 ($s^r = 0.7$) or 4 ($s^r = 0.3$) disjunctively combined elements S_s are created by canonical conversions, a non-canonical approach is favorable. Thus, our practical experiments verify our theoretical results and show their correctness even in case of a certain practical implementation.

Practical Analysis of Influences of Redundancy. In our theoretical analysis we have shown that predicate redundancy r_p does not influence the memory requirements of algorithms. However, in a practical realization this property does not hold. The influence of r_p on our implementation is illustrated in Fig. 4. Ordinates show an increasing num-

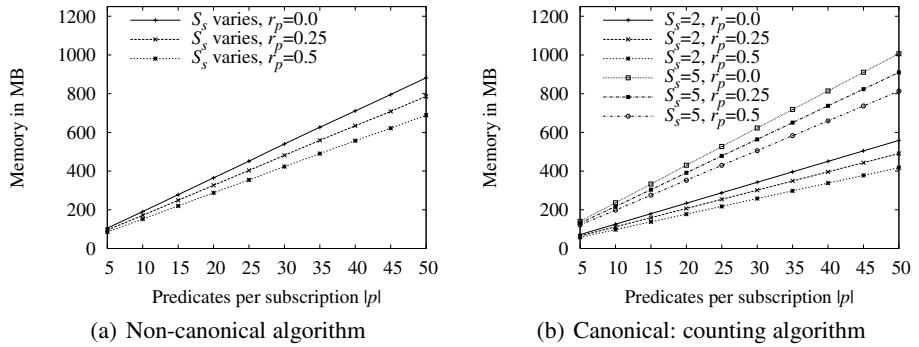


Fig. 4. Influence of predicate redundancy r_p on the algorithms in case of $|s| = 1,000,000$, $op^r = 0.5$ and $s^r = 0.3$

ber of predicates per subscription $|p|$, abscissae are labeled with the required memory in MB. In this experiment we have registered 1,000,000 subscriptions, further characterizing parameters are $op^r = 0.5$ and $s^r = 0.3$. The behavior of the non-canonical approach with varying predicate redundancy is shown in Fig. 4(a), the counting algorithm is presented in Fig. 4(b) for varying S_s and r_p .

Both algorithms show decreasing memory requirements with increasing r_p . This behavior results out of the decreasing memory overhead in a practical implementation: Both algorithms utilize a predicate subscription association table, which requires a dynamic implementation causing more memory usage. If there are less unique predicates, which is caused by predicate redundancy, the amount of memory overhead decreases. Thus, the total memory requirements decrease as observable in Fig. 4.

5.2 Practical Analysis of Efficiency

We are aware of the correlation between memory usage and filter efficiency of filtering algorithms: We cannot utilize the most space efficient algorithm in practice if it shows poor time efficiency. Vice versa, time efficient solutions, such as [7], might become inapplicable due to their memory requirements [3]. Thus, in our analysis we also compared the time efficiency of the counting (CNT) and the non-canonical approach (NCA) to confirm the applicability of the non-canonical approach in practice. In our experiments, we only have to compare the time efficiency of subscription matching, since predicate matching works the same in both algorithms. Time efficiency is represented by the average filtering time for subscription matching per event, i.e., increasing times denote decreasing efficiency. We ran our experiments several times to obtain negligible variances. Thus, in the figures we only show the mean values of filtering time.

Figure 5 shows the influence of the number of subscriptions registered with the pub/sub system. In Fig. 5(a), we have used $|p| = 10$, Fig. 5(b) illustrates time efficiency in case of $|p| = 30$. We show the behavior of the counting algorithm for the two cases $S_s = 4$ and $S_s = 8$. Predicate redundancy is chosen with $r_p = 0.0$ and $r_p = 0.5$. We also present the non-canonical approach assuming the worst case behavior, i.e., if

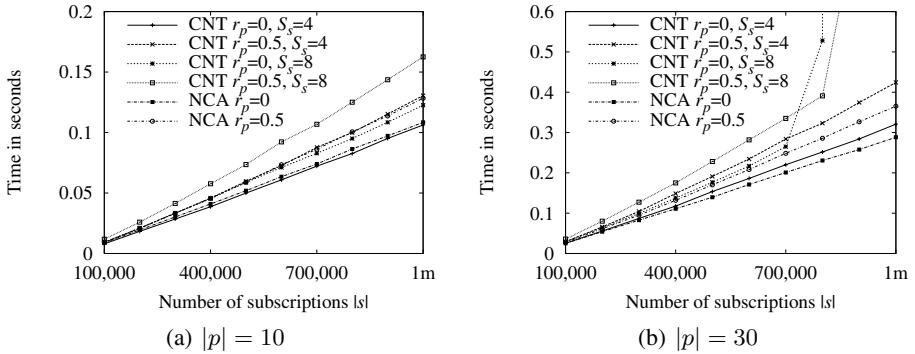


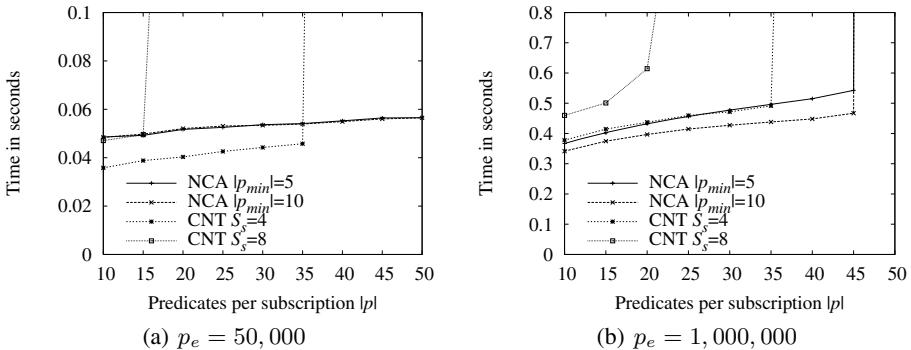
Fig. 5. Influence of number of subscriptions $|s|$ for varying r_p and S_s

a candidate subscription is evaluated, its whole Boolean expression is analyzed. Thus, we always test entire subscription trees in our experiments. In this experiment, we have increased the number of fulfilled predicates per event p_e with growing subscription numbers: $p_e = \frac{|s||p|}{50}$. We have chosen the minimum number of fulfilled predicates required for matching $|p_{min}|$ with 5 in case of $|p| = 10$ and with 10 in case of $|p| = 30$.

Figure 5 illustrates the average filtering times at the ordinates. Both algorithms show linearly increasing filtering times in case of growing subscription numbers. In case of $S_s = 8$ and $|p| = 30$ (Fig. 5(b)), the counting algorithm requires more memory than the available resources (sharp bends in curves). Thus, the operation system starts page swapping resulting in strongly increasing filtering times in case of more than 700,000 and 800,000 subscriptions (according to r_p , cf. Sect. 5.1). Generally, increasing predicate redundancy r_p leads to growing filtering times for both algorithms in the evaluated setting. This is due to the fact that more candidate subscriptions have to be evaluated (non-canonical algorithm) and more counters have to be increased in the hit vector (both algorithms). The counting algorithm in case of $S_s = 8$ always shows the worst time efficiency. According to the number of predicates $|p|$, either the non-canonical approach (Fig. 5(b)) or the counting algorithm with $S_s = 4$ (Fig. 5(a)) are the most efficient filtering approaches (nearly on par with the other approach).

The influence of $|p|$ is shown in Fig. 6. In Fig. 6(a), it holds $p_e = 50,000$, Fig. 6(b) shows the case of $p_e = 1,000,000$. For the non-canonical approach we analyzed the two settings $|p_{min}| = 5$ and $|p_{min}| = 10$. The counting approach is presented in two variants with $S_s = 4$ and $S_s = 8$. We have registered 1,000,000 subscriptions.

Again, sharp bends in the curves in Fig. 6 denote the point of exhausted main memory resources. The non-canonical approach shows the best scalability, followed by the counting approach in case of $S_s = 4$. We can also observe improved time efficiency in the non-canonical approach in case of higher $|p_{min}|$. This effect becomes more apparent with a high value of p_e (Fig. 6(b)) due to more candidate subscriptions requiring evaluation. In case of a small number of fulfilled predicates per event p_e , the counting (case $S_s = 4$) is more efficient than the non-canonical algorithm; large numbers of p_e clearly favor the non-canonical approach. The reason is the increased number of hits (incrementing the hit vector) in the counting approach due to canonical conversions.

**Fig. 6.** Influence of number of predicates $|p|$ for varying p_e

Our efficiency analysis shows that counting and non-canonical approach perform similarly for increasing problem sizes. In some cases, the counting approach shows slightly better time efficiency, other settings favor the non-canonical approach. For large S_s , the non-canonical approach shows both better time and space efficiency. Thus, a non-canonical solution offers better scalability properties in these situations.

6 Conclusions and Future Work

In this paper, we have presented a detailed investigation of two classes of event filtering approaches: canonical and non-canonical algorithms. As a first step we introduced a characterization scheme for qualifying primitive subscriptions in order to allow for a description of various practical settings. Based on this scheme, we thoroughly analyzed the memory requirements of three important event filtering algorithms (counting [1, 12], cluster [6, 8] and non-canonical [3]). We compared our results to derive conclusions about the circumstances under which canonical algorithms should be preferred in respect to memory usage and which settings favor non-canonical approaches.

To show the applicability of our theoretical results in a practical implementation, we proposed an implementation and investigated its memory requirements by experiment. This practical evaluation clearly verified our theoretical results: Even when conversions to canonical forms result in only two canonical subscriptions (i.e., subscription use only one disjunction), a non-canonical approach is favorable.

We also correlated the memory requirements of the practically analyzed algorithms to their filter efficiency. Generally, non-canonical algorithms show approximately the same time efficiency as canonical ones. In case of increasing numbers of disjunctions in subscriptions, the time efficiency of non-canonical approaches improves compared to canonical solutions. In this case, a non-canonical approach also shows much better scalability properties as demonstrated in our analysis of memory requirements. Thus, if subscriptions involve disjunctions, non-canonical algorithms are the preferred class of filtering solutions due to their direct exploitation of subscriptions in event filtering.

For future work, we plan to describe different application scenarios using our characterization scheme. A later analysis of these scenarios will allow conclusions about the preferred filtering algorithm for these applications. We also plan to further extend the non-canonical filtering approach to a distributed algorithm.

References

1. G. Ashayer, H. A. Jacobsen, and H. Leung. Predicate Matching and Subscription Matching in Publish/Subscribe Systems. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '02)*, Vienna, Austria, July 2–5 2002.
2. S. Bittner and A. Hinze. Investigating the Memory Requirements for Publish/Subscribe Filtering Algorithms. Technical Report 03/2005, Computer Science Department, University of Waikato, May 2005.
3. S. Bittner and A. Hinze. On the Benefits of Non-Canonical Filtering in Publish/Subscribe Systems. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '05)*, pages 451–457, Columbus, USA, June 6–10 2005.
4. A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient Filtering in Publish-Subscribe Systems using Binary Decision Diagrams. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001)*, Toronto, Canada, May 2001.
5. A. Carzaniga and A. L. Wolf. Forwarding in a Content-Based Network. In *Proceedings of the 2003 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '03)*, Karlsruhe, Germany, March 2003.
6. F. Fabret, A. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. In *Proceedings of the 2001 ACM SIGMOD*, pages 115–126, Santa Barbara, USA, May 21–24 2001.
7. J. Gough and G. Smith. Efficient Recognition of Events in a Distributed System. In *Proceedings of the 18th Australasian Computer Science Conference*, Adelaide, Australia, 1995.
8. E. N. Hanson, M. Chaabouni, C.-H. Kim, and Y.-W. Wang. A Predicate Matching Algorithm for Database Rule Systems. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data (SIGMOD 1990)*, Atlantic City, USA, May 23–25 1990.
9. G. Mühl and L. Fiege. Supporting Covering and Merging in Content-Based Publish/Subscribe Systems: Beyond Name/Value Pairs. *IEEE DSOnline*, 2(7), 2001.
10. F. Peng and S. S. Chawathe. XPath Queries on Streaming Data. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD 2003)*, pages 431–442, San Diego, USA, June 9–12 2003.
11. B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of the Australian UNIX and Open Systems User Group Conference (AUUG97)*, Brisbane, Australia, September 3–5 1997.
12. T. W. Yan and H. García-Molina. Index Structures for Selective Dissemination of Information Under the Boolean Model. *ACM Transactions on Database Systems*, 19(2), 1994.

Mapping Discovery for XML Data Integration

Zoubida Kedad and Xiaohui Xue

Laboratoire PRISM, Université de Versailles,
45 avenue des Etats-Unis, 78035 Versailles, France
`{Zoubida.kedad, Xiaohui.Xue}@prism.uvsq.fr`

Abstract. The interoperability of heterogeneous data sources is an important issue in many applications such as mediation systems or web-based systems. In these systems, each data source exports a schema and each application defines a target schema representing its needs. The way instances of the target schema are derived from the sources is described through mappings. Generating such mappings is a difficult task, especially when the schemas are semi structured. In this paper, we propose an approach for mapping generation in an XML context; the basic idea is to decompose the target schema into subtrees and to find mappings, called partial mappings, for each of them; the mappings for the whole target schema are then produced by combining the partial mappings and checking that the structure of the target schema is preserved. We also present a tool supporting our approach and some experimental results.

1 Introduction

A broad variety of data is available on the Web in distinct heterogeneous sources. The exchange and integration of these data sources is an important issue in many applications such as mediation systems or web-based systems.

In these systems, each data source has a schema (called source schema) that presents its data to the outside world. Applications needs are represented by target schemas. The way instances of the target schema are derived from instances of the source schemas is described through mappings. One example of systems using these mappings is mediation systems, where the target schema is called mediation schema and the mappings are called mediation queries. The user queries are expressed over the mediation schema and rewritten in terms of the source schemas using the mappings.

Defining mappings is a difficult task which requires a deep understanding not only of the semantics of the source schemas, but also the semantic links between the sources and the target schema. The complexity of this task increases when the number of data sources is high. The amount of required knowledge makes the manual definition of the mappings extremely difficult for a human designer. When the target schema and the source schemas are in XML, the definition of the mappings is more complex because of the hierarchical nature of the data.

In [7], we have proposed a general framework for mapping generation. In this paper, we present the algorithms for automatic mapping generation and a tool to support this task. We consider that the target and source schemas are described in XML Schema, and we assume that a set of correspondences is provided. These correspondences relate elements of a source and elements of the target schema and express that

these elements represent the same concept. Our tool produces a set of mappings, corresponding to different ways to derive instances of the target schema from instances of the sources. The generated mappings can be expressed in a standard language, such as XQuery or XSLT.

Due to the semi-structured nature of XML sources, it is extremely difficult to directly define mappings for the whole target schema. The basic idea of our approach is (i) firstly to decompose the target schema into a set of subtrees, called **target subtrees**; (ii) then to find the different ways, called **partial mappings**, to define each target subtree from the source schemas; (iii) and finally to combine the partial mappings to generate the mappings for the whole schema, called **target mappings**.

The paper is organized as follows. In Section 2, we give some basic assumptions and preliminary definitions. Section 3 presents the decomposition of the target schema. Section 4 and Section 5 detail the determination of the partial mappings and the generation of the target mappings respectively. Section 6 gives some experimental results obtained by our system. Some related works are presented in Section 7 and Section 8 concludes the paper.

2 Preliminaries

In this section we present the underlying assumptions of our approach: the representation of the target and the source schemas, and the correspondences between the schemas.

2.1 Representation of Target and Source Schemas

We consider source schemas and target schema expressed using XML Schema. Figure 1 shows two source schemas and a target schema representing information about books in a library. To avoid confusions, in the rest of the paper, each node will be suffixed by the name of its schema: $AuthorId_{s1}$ will refer to the node $AuthorId$ in S1 while $ISBN_{s2}$ will refer to the node $ISBN$ in S2. Every node in the tree may be either a text node (e.g. $AuthorId_{s1}$), that is, a node containing only text, or an internal node (e.g. $Chapter_{s1}$). The leaf nodes of the tree are always text nodes.

The cardinality of every node is characterized by the attributes `minOccur` and `maxOccur`, representing respectively the minimum and maximum number of instances for this node in the tree with respect to its parent. Each node is monovalued (`maxOccurs = 1`) or multivalued (`maxOccurs > 1`); it is also optional (`minOccurs = 0`) or mandatory (`minOccurs > 0`). In Figure 1, the symbol ‘+’ represents a multivalued and mandatory node (e.g. $Book_{s2}$); the symbol ‘*’ represents a multivalued and optional node (e.g. $Book_{ts}$); and the symbol ‘?’ represents a monovalued and optional node (e.g. $Abstract_{ts}$). A node without symbol is monovalued and mandatory (e.g. Id_{s1}).

Keys are defined either in the whole schema or only in a subtree of the schema. In the first case, the key is absolute. In the second case, the key is relative and its scope is an antecessor of the identified node, except the root. In Figure 1, the nodes written in bold represent keys. If the name of the key node is followed by a bracket, then the key is a relative key and its scope is the node between brackets (e.g. $Number_{s2}$ is a

relative key and its scope is $Book_{s2}$), otherwise it is an absolute key (e.g. $ISBN_{s1}$). A schema may also contain references; each one is a set of text nodes referencing another set of text nodes defined as a key. In our example, $AuthorId_{s1}$ references Id_{s1} , and this is represented by an arrow in Figure 1.

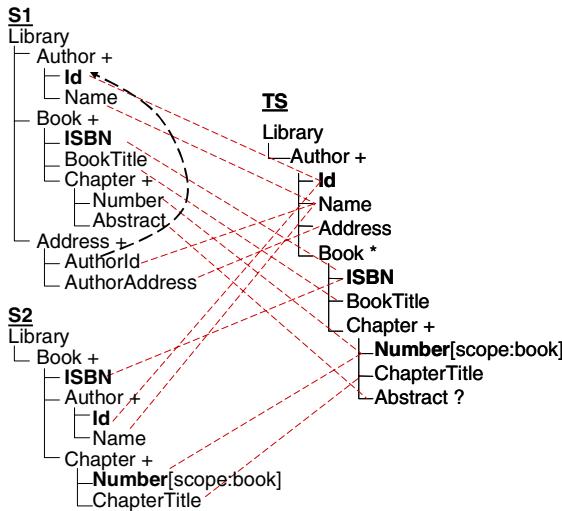


Fig. 1. Schemas and correspondences

2.2 Semantic Correspondences

We suppose that a set of semantic correspondences is provided between each source schema and the target schema. The definition of these correspondences is an important issue and several approaches have been proposed to solve this problem [4][5][12].

In our work, we consider two kinds of correspondences: 1-1 and 1-n. A 1-1 correspondence relates a target node n with a source node n' , and states that the two nodes represent the same concept. This correspondence is denoted $n \cong n'$ (e.g. $Id_{s1} \cong Id_{ts}$, $Number_{s2} \cong Number_{ts}$). In Figure 1, dotted lines represent correspondences. A transformation function may be applied to the source node. For example, a correspondence can be specified to relate a target node $PriceInEuro$ to a source node $PriceInDollar$; if the exchange rate is $1\text{€} = 0,797\text{\$}$, such correspondence is denoted $PriceInEuro \cong 0,797 * PriceInDollar$.

A 1-n correspondence relates a target node n to a set of source nodes combined by the mean of a transformation function. For example, a target node $Name$ represents the same concept as the concatenation of two source nodes $FirstName$ and $LastName$. This correspondence is denoted $Name \cong \text{concat}(FirstName, LastName)$.

More generally, we consider the correspondences relating a target node n and a set of source nodes n_1, \dots, n_k combined using a function f . Such correspondences are

denoted $n \cong f(n_1, \dots, n_k)$. For simplicity, in this paper we will restrict ourselves to 1-1 correspondences.

We use the same notation to represent correspondences between sets of nodes. There is a correspondence between two sets of nodes s_1 and s_2 if (i) both s_1 and s_2 contain the same number of nodes (ii) and for each node n_1 in s_1 there is exactly one node n_2 in s_2 such that $n_1 \cong n_2$, and vice versa. The correspondence between the two sets s_1 and s_2 is denoted $s_1 \cong s_2$ (e.g. $\{ISBN_{s_1}, BookTitle_{s_1}\} \cong \{ISBN_{s_2}, BookTitle_{s_2}\}$).

Correspondences between two source schemas are derived through their correspondences with the target schema. Given two source nodes n and n' in S and S' respectively, the correspondence $n \cong n'$ holds if there is a node n'' in the target schema such that $n'' \cong n$ and $n'' \cong n'$. Some correspondences may also be provided between the source schemas; they will be used in our approach for mapping generation.

3 Decomposing the Target Schema

To handle the complexity of mapping definition, we decompose the target schema into a set of subtrees, called target subtrees; we will first find mappings for each target subtree then combine these mappings to generate the mappings for the whole schema, called target mappings.

Given a target schema, each target subtree t is a subtree of the target schema satisfying the following conditions:

- the root r of the subtree is either a multivalued node or the root of the target schema;
- all the other nodes in t are descendants of r and are monovalued;
- there is at least one text node in t (t may contain a single node).

This decomposition of the target schema gives several subtrees in which every node is monovalued. The mapping generation problem for the target schema is decomposed into two steps: finding mappings for every target subtree, then combining these mappings. Since a target subtree contains only monovalued nodes except the root, finding a mapping for this subtree consists in finding some equivalent nodes in the sources that satisfy the cardinalities constraints regardless their hierarchical organization. The hierarchical structure of the different target subtrees is checked during the second step.

Our target schema given in Figure 1 has three target subtrees shown on the right side of Figure 2: t_1 is composed of the multivalued node $Author_{ts}$ and its three monovalued children Id_{ts} , $Name_{ts}$ and $Address_{ts}$; t_2 is composed of $Book_{ts}$ and its two monovalued children $ISBN_{ts}$ and $BookTitle_{ts}$; and t_3 is composed of $Chapter_{ts}$, $Number_{ts}$, $ChapterTitle_{ts}$ and $Abstract_{ts}$. The root $Library_{ts}$ doesn't belong to any target subtree.

Given two target subtrees t and t' such that the root of t' is a child of a node in t , we say that t is the parent of t' and t' is a child of t (e.g. in Figure 2, t_2 is the child of t_1 and the parent of t_3).

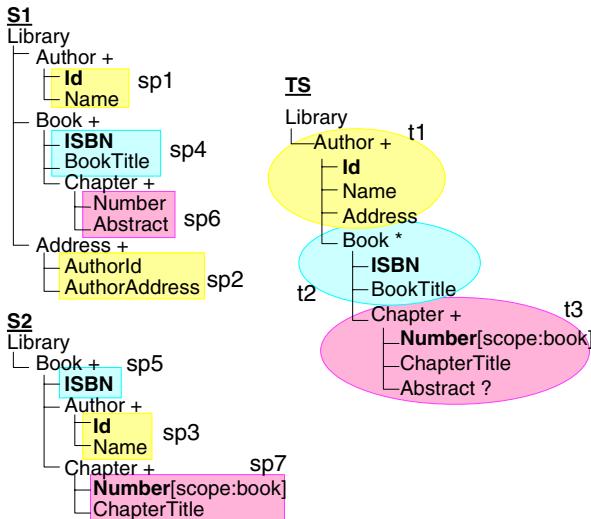


Fig. 2. Target subtrees and source parts

A target subtree can be either mandatory or optional in a target schema. Consider a target schema with the root R and a subtree t of this schema with the root r . If t has a parent subtree t' with the root node r' , we say that t is mandatory if all the nodes on the path from r to r' (except r') are mandatory. If t has no parent subtree, it is mandatory if all the nodes on the path from r to R are mandatory. In all the other cases, t is optional. In our example, t_1 and t_3 are mandatory and t_2 is optional.

4 Determining Partial Mappings

Each partial mapping represents a way to derive instances of a target subtree from the instances of the source schemas. The partial mappings of a given target subtree are determined independently from the other subtrees in three steps: (i) identifying the parts of the sources (called **source parts**) that are relevant for the considered target subtree; (ii) searching the joins to combine these source parts; (iii) and determining the partial mappings from the source parts and the joins between them. Each target subtree may have several partial mappings with different semantics. In the rest of this section, we will describe these three steps.

4.1 Identifying Source Parts

A source part of a given target subtree is a set of text nodes in the source schemas that can contribute to derive instances for this target subtree.

Before defining source parts, we first present an extended definition of node cardinality. In XML Schema, the cardinality of a node is given with respect to the parent node: a node is multivalued or monovalued with respect to its parent. We generalize this definition to any pair of nodes.

Def. 1. Extended definition of Cardinality. Given two nodes n and n' in a schema and their first common antecessor m , n is monovalued with respect to n' if every node on the path from m to n (except m) is monovalued. Otherwise, n is multivalued with respect to n' .

According to the definition, $ISBN_{s1}$ is monovalued with respect to $BookTitle_{s1}$: their common antecessor is $Book_{s1}$ and the only node on the path from $Book_{s1}$ to $ISBN_{s1}$ (except $Book_{s1}$) is $ISBN_{s1}$, which is monovalued. Similarly, $BookTitle_{s1}$ is monovalued with respect to $ISBN_{s1}$. $Number_{s1}$ is multivalued with respect to $ISBN_{s1}$ because their common antecessor is $Book_{s1}$ and the path from $Book_{s1}$ to $Number_{s1}$ contains $Chapter_{s1}$ which is multivalued. On the contrary, $ISBN_{s1}$ is monovalued with respect to $Number_{s1}$.

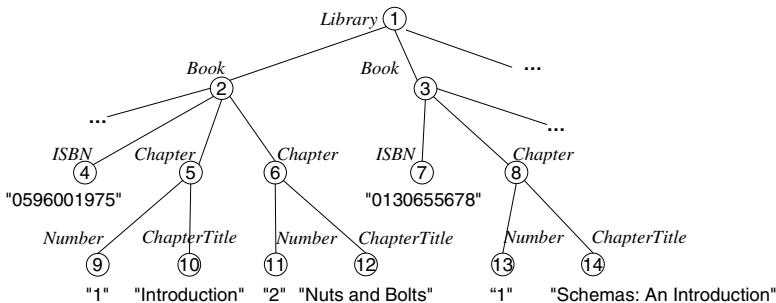


Fig. 3. An example of instances for the source S2

Note that this extended definition of cardinality is different from the definition of functional dependency. Consider the nodes $ChapterTitle_{s2}$ and $Number_{s2}$ in S2. $ChapterTitle_{s2}$ is monovalued with respect to $Number_{s2}$. However, the functional dependency $Number_{s2} \rightarrow ChapterTitle_{s2}$ doesn't hold as we can see in Figure 3: two different instances of chapter number may have the same value, but associated with different titles; in fact, there are several titles for a given chapter number, one for each book.

Given a target subtree t , a source part sp for t in the source schema S is a set of text nodes that satisfies the following conditions:

- there is a set of text nodes c in t such that $c \equiv sp$;
- there is at least one node n in sp such that the other nodes in sp are monovalued with respect to n ;
- except c , there is no set of text nodes c' in S such that $sp \subseteq c'$ and c' satisfies the two above conditions.

Given a target subtree t , every source node involved in a correspondence with the nodes of t is found in at least one source part for t . If no source part is found for a target subtree, this means that there is no correspondent node in the sources for any of the nodes of this target subtree.

Consider the target subtree t_1 having the text nodes Id_{ts} , $Name_{ts}$ and $Address_{ts}$. These nodes have the corresponding nodes Id_{s1} , $Name_{s1}$, $AuthorId_{s1}$ and $AuthorAddress_{s1}$ in S1. In the set $\{Id_{s1}, Name_{s1}\}$, both Id_{s1} and $Name_{s1}$ are monovalued with

respect to the other; this set is therefore a source part for t1. In $\{AuthorId_{s1}, AuthorAddress_{s1}\}$, both $AuthorId_{s1}$ and $AuthorAddress_{s1}$ are monovalued with respect to the other; this set is therefore a source part for t1. $\{Id_{s1}\}$ is not a source part because it is a subset of $\{Id_{s1}, Name_{s1}\}$. $\{AuthorId_{s1}, AuthorAddress_{s1}, Id_{s1}\}$ is not a source part also because Id_{s1} is multivalued with respect to both $AuthorId_{s1}$ and $AuthorAddress_{s1}$ and both $AuthorId_{s1}$ and $AuthorAddress_{s1}$ are multivalued with respect to Id_{s1} .

The source parts for the target subtrees of our running example are shown on the left side of Figure 2. The subtree t1 has two source parts sp1 and sp2 in S1 and one source part sp3 in S2; t2 has two source parts sp4 and sp5 in S1 and S2 respectively; and t3 has two source parts sp6 and sp7.

4.2 Identifying Join Operations

The joins between source parts are identified using keys and key references. There are two distinct cases: the two source parts either belong to the same source schema or to different ones.

Given two source parts sp and sp' in the same source schema, a join is possible if there are two sets of text nodes c and c' in the schema such that:

- c is a key and c' references c;
- there is a node n in c such that every node in sp is monovalued with respect to n;
- there is a node n' in c' such that every node in sp' is monovalued with respect to n'.

In this case, a join is possible between sp and sp' with the join predicate $c = c'$; it is denoted $j[c = c'](sp, sp')$. For example, the join $j[Id_{s1} = AuthorId_{s1}](sp1, sp2)$ is possible between sp1 and sp2 since $AuthorId_{s1}$ is a reference on Id_{s1} .

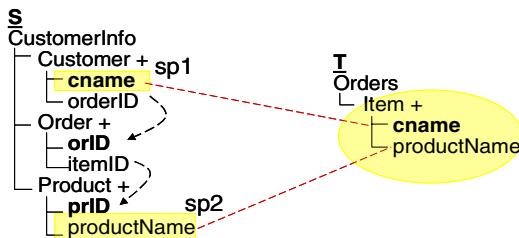


Fig. 4. Relating two source parts through several references

This definition can be generalized by considering a sequence of references from c' to c instead of a single one. Consider the example shown in Figure 4. In the source S, two source parts sp1 and sp2 correspond to the single subtree of the target schema and no join is possible between them using the previous rule because no reference relates them directly. However, they are related through the two references: $orderID_s$

referencing $orID_s$ and $itemID_s$ referencing $prID_s$. A join is therefore possible and it is denoted $j[orderID_{s1} = orID_{s1}, itemID_{s1} = prID_{s1}](sp1, sp2)$.

A join can also be possible between sources parts of different schemas. Consider two source parts sp and sp' in the source schemas S and S' respectively. Given a set of text nodes c in S and a set of text nodes c' in S' , a join can be applied to sp and sp' with the predicate $c = c'$ if the following conditions hold:

- $c \cong c'$;
- either c or c' is an absolute key in its schema;
- there is a node n in c such that every node in sp is monovalued with respect to n ;
- there is a node n' in c' such that every node in sp' is monovalued with respect to n' .

In our example, the join $j[Id_{s1} = Id_{s2}](sp1, sp3)$ is possible between $sp1$ and $sp2$ because both Id_{s1} and Id_{s2} are defined as absolute keys. The join between $sp6$ and $sp7$ with the predicate $Number_{s1} = Number_{s2}$ is not possible because neither $Number_{s1}$ nor $Number_{s2}$ is defined as an absolute key. However, we know that the combination $\{Number_{s2}, ISBN_{s2}\}$ is unique in the whole schema because the scope of $Number_{s2}$ is $Book_{s2}$ which has $ISBN_{s2}$ as the absolute key $ISBN_{s2}$. We have therefore to consider the combination $\{Number_{s2}, ISBN_{s2}\}$ as an absolute key and use it instead of $Number_{s2}$. In fact, each time a relative key is found, it is combined with other key nodes to get an absolute key if possible. Figure 5 shows all the possible joins in our example.

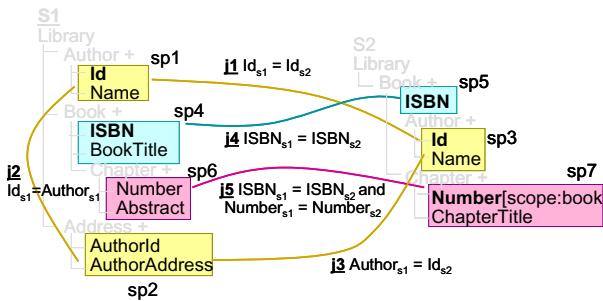


Fig. 5. Join operations

In our approach, we consider that a join is possible in a limited number of cases; we do not therefore generate all the possible joins but only a subset of them. For example, a join involving two different sources is considered as possible only if the join predicate involves an absolute key. We could also have considered that a join is possible each time a correspondence is found between two sets of nodes, regardless the key definitions. But in our opinion, the semantics of this operation is not clear and we therefore do not consider these joins. Consequently, only a subset of all the possible target mappings is generated in our approach.

4.3 Defining Partial Mappings from the Source Parts and the Joins

The partial mappings of a target subtree are determined using the corresponding source parts and the joins between them.

The source parts and the joins corresponding to a given target subtree are represented by a graph called join graph where every node is a source part and every edge between two source parts is a join between them; the edges are numbered and labeled with the join predicate.

Given the join graph G for a target subtree t , each partial mapping for t , denoted pm , is defined as a connected acyclic sub-graph of G such that for every mandatory text node n in t , there is at least a node n' in one of its source parts such that $n \equiv n'$.

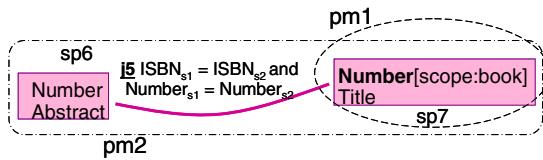


Fig. 6. Join graph of t_3

```

Partial_Mapping_Determination(G(SP, J), st, PM)
Begin
    PM := Ø;
    for each source part sp in SP:
        J' := Ø;
        SP' := {sp};
        Build_Partial_Mapping (G'(SP', J'), G(SP, J), st, PM);
    return (PM);
End

Build_Partial_Mapping (G'(SP', J'), G(SP, J), st, PM)
Begin
    if mandatory_text_nodes(SP', st) //returns true if SP' contains all the mandatory text nodes in st
    then
        PM := PM ∪ {G'(SP', J')};
        //adds a new partial mapping represented by the graph G' to set PM
        for each join j between the source parts sp and sp' such that sp' ∈ SP' and sp ∈ SP'
            SP'' := SP' ∪ {sp};
            J'' := J' ∪ {j};
            if G''(SP'', J'') ∈ PM
                //adding the edge representing the join j to the subgraph G' doesn't give an element of PM
                then
                    Build_Partial_Mapping (G''(SP'', J''), G(SP, J), st, PM);
    End

```

Fig. 7. The algorithm of partial mapping determination

Considering the subtree t_3 of Figure 2, the corresponding join graph is shown in Figure 6. It contains two source parts sp_6 , sp_7 and the join j_5 . In this graph, there are two partial mappings: pm_1 containing a single source part sp_7 and pm_2 containing sp_6 and sp_7 related by j_5 . Both are connected acyclic sub-graphs of the join graph and both produce instances for the mandatory text nodes $Number_{ts}$ and $ChapterTitle_{ts}$ in t_3 ; pm_1 does not produce instances for the optional node $Abstract_{ts}$; pm_2 joins the two

source parts; it may produce fewer chapters than pm1 but more information for every chapter (its abstract).

For simplicity, in the rest of the paper, we refer to a partial mapping by the source part name if it contains a single source part, or by the names of the corresponding joins if it contains more than one source part. In our example, pm1 and pm2 are denoted {sp7} and {j5} respectively.

The algorithm for partial mapping determination is given in Figure 7. It is a recursive algorithm that takes as input one target subtree (st) and the corresponding join graph $G(SP, J)$ where SP represents the set of nodes (the source parts) and J the set of edges (the possible joins). The algorithm produces the set of all the partial mappings (PM) for st ; each partial mapping in PM is represented by the corresponding sub-graph.

5 Generating Target Mappings

The mappings for the whole target schema, called target mappings, are defined using the partial mappings. To perform this task, **candidate mappings** are first generated by combining the partial mappings of the different target subtrees. Then the parent-child relations between the target subtrees are checked to produce target mappings.

A candidate mapping cm for the target schema TS is a set of partial mappings such that:

- there is at most one partial mapping for each target subtree in TS ;
- for each mandatory target subtree t having no parent subtree, there is one partial mapping for t ;
- for each mandatory subtree t having the parent subtree t' , if there is a partial mapping for t' then there is also a partial mapping for t , and vice versa.

Consider the following partial mappings in our example: $pm3 = \{j2\}$ and $pm4 = \{j1, j2\}$ for $t1$; $pm5 = \{j4\}$ for $t2$; and $pm2 = \{j5\}$ for $t3$. Since $t2$ is optional and its child $t3$ is mandatory, each candidate mapping denoted cm_i either contains no partial mapping for both $t2$ and $t3$ such as $cm1 = \{pm3\}$ and $cm2 = \{pm4\}$, or contains a partial mapping for both $t2$ and $t3$ such as $cm3 = \{pm3, pm5, pm2\}$ and $cm4 = \{pm4, pm5, pm2\}$.

The algorithm for candidate mapping generation is given in Figure 8. This algorithm takes as input the target schema TS and the sets of partial mappings PM_1, \dots, PM_n corresponding respectively to the subtrees t_1, \dots, t_n in TS . The algorithm performs a top-down browsing of the subtrees in TS and generates the set of candidate mappings CM .

Target mappings are derived from the candidate mappings that satisfy the parent-child relations between the target subtrees. Consider a target subtree t , its parent subtree t' and their respective partial mappings pm and pm' ; pm and pm' preserve the parent-child relation between t and t' if the following conditions hold:

- there is a source part sp in pm and a source part sp' in pm' which are in the same source;
- there is either a node in sp with respect to which all the nodes in sp' are monovalued; or a node in sp' with respect to which all the nodes in sp are monovalued.

```

Candidate_Mapping_Generation(TS, PM1, ..., PMn, CM)
Begin
    CM := Ø; // each element of CM is a set of partial mappings
    for each target subtree ti in get-mandatory-top-subtrees(TS)
        // get-mandatory-top-subtrees(TS) returns the subtrees in TS that are mandatory and has not parent subtrees
        if PMi == Ø
            then return (Ø);
        //if a mandatory top subtree has no partial mapping, then the target schema has no target mapping
        if CM == Ø
            then
                for each partial mappings pm in PMi
                    CM := CM ∪ {pm};
            else
                for each set S in CM
                    for each partial mapping pm in PMi
                        S' := S ∪ {pm};
                        CM := CM ∪ {S'};
                        CM := CM - {S};
        for each target subtree ti in TS not in get-mandatory-top-subtrees(TS) from top to down:
            if (top(ti)) // returns true if ti has not parent subtree
                for each set S in CM
                    for each partial mapping pm in PMi
            else
                for each set S in CM
                    if contains_parent_mapping(ti, S)
                        //returns true if the set S contains a partial mapping for the parent subtree of ti
                        then
                            for every pm in PMi
                                S' := S ∪ {pm};
                                CM := CM ∪ {S'};
                            if (mandatory(ti)) then CM := CM - {S};
        return (CM);
End

```

Fig. 8. The algorithm of candidate mapping generation

If there is a node in sp with respect to which all the nodes in sp' are monovalued, then for every instance of sp we can find the corresponding instance of sp' , and for every instance of sp' we can find the corresponding instances of sp . The parent-child relation is therefore satisfied.

For the target schema of our example, there are two parent-child relations to check: one between t_1 and t_2 and the other between t_2 and t_3 .

Consider the candidate mapping $cm4 = \{pm4, pm5, pm2\}$. The parent-child relation between t_1 and t_2 is satisfied in $cm4$ because every node in sp_5 (involved in $pm5$) is monovalued with respect to both Id_{s2} and $Name_{s2}$ in sp_3 (involved in $pm4$). The parent-child relation between t_2 and t_3 is also satisfied because every node in sp_5 is monovalued with respect to both $Number_{s2}$ and $ChapterTitle_{s2}$ in sp_7 (in $pm2$). Therefore, $cm4$ is a target mapping for TS .

The candidate mappings $cm1$ and $cm2$ are also target mappings because both contain a single partial mapping. The candidate mapping $cm3$ does not lead to a target mapping because the parent-child relation between t_1 and t_2 is not satisfied.

Other target mappings can be derived by applying set-based operations like Union, Intersection and Difference to two or more mappings. For example the union of $cm1$ and $cm4$ is a new target mapping that takes the union of $pm4$ and $pm3$ for t_1 , $pm5$ for t_2 and $pm2$ for t_3 .

Each target mapping is an abstract query that can be translated into a specific query language such as XQuery or XSLT. To translate a target mapping into XQuery, each partial mapping is translated into a FWR (For-Where-Return) expression. For each

```

<Library>{
    for $au in distinct(S1/Library/Author/Id, S1/Library/Address/AuthorId, S2/Library/Book/Author/Id)
    for $sp1 in S1/Library/Author
    for $sp2 in S1/Library/Address
    for $sp3 in S2/Library/Book/Author
    where $sp1/Id=$sp2/AuthorId and $sp1/Id=$sp3/Id and $sp1/Id=$au
    return <Author>{
        <Id>(data($sp1/Id))</Id>,
        <name>(data($sp1/Name))</name>,
        <Address>(data($sp2/AuthorAddress))</Address>
        for $b in distinct(S2/Library/Book/ISBN, S1/Library/Author/Book/ISBN)
        for $sp4 in S1/Library/Author/Book
        for $sp5 in S2/Library/Book[Author/Id = sp3/Id]
        where $sp4/ISBN = $sp5/ISBN and $sp4/ISBN = $b
        return <Book>{
            <ISBN>(data($sp4/ISBN))</ISBN>,
            <BookTitle>(data($sp4/title))</BookTitle>
            for $c in distinct(S1/Library/Author/Book/Chapter/Number, S2/Library/Book/Chapter/Number)
            for $sp6 in S1/Library/Author/Book/Chapter
            for $sp7 in S2/Library/Book[ISBN=sp5/ISBN]/Chapter
            where $sp6/Number = $sp7/Number and $sp7/Number = $c
            return <Chapter>{
                <Number>(data($sp6/ISBN))</Number>,
                <ChapterTitle>(data($sp7/ChapterTitle))</ChapterTitle>,
                <Abstract>(data($sp6/Abstract))</Abstract>
            }</Chapter>
        }</Book>
    }</Author>
}</Library>

```

Fig. 9. An XQuery target mapping

target subtree t and its parent t' , the FWR expression of t is nested in the FWR expression of t' . A grouping operation is added for every key in the target schema. For example, Figure 9 gives the translation to XQuery of the target mapping cm4.

6 Experimental Results

We implemented a system [8] in Java and we have run five scenarios to evaluate its performance. Table 1 summarizes the main characteristics of these scenarios, such as the number of nodes in the target schema, the number of data sources and the number of nodes for each one, the number of correspondences between the sources and the target schema, and the number of the key definitions in the sources.

The first scenario is from the Mediagrid project¹ which proposes a mediation framework for a transparent access to biological data sources; it considers three biological sources SGD, GOLD, SMD and a target schema built by domain experts. The Library1 scenario contains six source schemas. The Library2 scenario is similar to Library1 but the overlap between the sources is more important (more correspondences are defined for the same number of text nodes in the target schema). The ABC1 and ABC2 scenarios contain 50 source schemas. They are similar, except that the ABC2 scenario contains 58 key definitions while ABC1 contain no key definitions.

¹ Supported by the French Ministry of Research through the ACI Grid program, www-lsr.imag.fr/mediagrid/.

Table 1. Characterizing the scenarios

Scenarios	Target schema			Correspondences	Source schemas				
	Depth	Nodes	Text nodes		Schemas	Nodes	Text nodes	Keys	Refs
Mediagrid	6	18	12	22	3	1674	825	412	413
Library1	5	18	14	26	6	56	30	9	1
Library2	5	18	14	30	6	62	35	10	1
ABC1	7	47	36	1300	50	1650	1434	0	0
ABC2	7	47	36	1300	50	1650	1434	58	0

We have run these different scenarios on a PC-compatible machine, with a 2.8G Hz P4 CPU and 516MB RAM, running Windows XP and JRE1.4.1. Each experiment is repeated five times and the average of the five is used as the measurement.

Table 2. Measuring the scenarios

Scenarios	Execution time (s)			
	Load	Target Schema Decomposition	Partial Mapping Determination	Target Mapping Generation
Mediagrid	1.44	0.001	0.02	0.002
Library1	0.44	0.001	0.067	0.095
Library2	0.046	0.001	0.105	0.25
ABC1	0.98	0.001	0.06	1.997
ABC2	1.03	0.001	316	27

The time needed for the main steps of our approach using the different scenarios are shown in Table 2. The loading time indicates the time to read the schemas and the correspondences into our internal representation. As expected, it is correlated to the size of the schemas and the number of their correspondences.

The target schema decomposition time indicates the time to decompose the target schema into target subtrees. We can see that the time needed to perform the task is negligible.

The partial mapping determination (pmd) time is proportional to the number of correspondences between target nodes and source nodes and the key and key references in the sources. The pmd time for Library1 which has 26 correspondences is smaller than the one of Library2 which has 30 correspondences; the two scenarios have the same number of sources and the same target schema. The pmd time for the ABC2 scenario which has 58 keys is largely greater than the one of the ABC1

scenario. This is because the number of keys of the ABC2 scenario makes the join graph very complex.

The target mapping generation (tmg) time indicates the time to find all the candidate mappings and to generate the target mappings. The tmg time is greater in ABC2 than in the other scenarios because in ABC2, most of the target subtrees have a lot of partial mappings (about 150), which leads to much more combinations to consider.

Some evaluations for the partial mapping determination and the target mapping generation are shown in Figure 10. The pmd time is decomposed into source part identification time, join identification time and partial mapping determination time. Figure 10 (a) shows the source part identification time with respect to the number of semantic correspondences between the target schema and the source schemas. The measures are done using the ABC2 scenario and considering 52 to 1300 correspondences. This task is proportional to the number of correspondences and its time is almost negligible (about only 0.022 second for 1300 correspondences).

Figure 10 (b) shows the time of join identification with respect to both the number of key definitions and the number of correspondences on the keys. We have considered the ABC2 scenario and we have successively increased both the number of key definitions and the number of correspondences involving keys. The time needed to perform this task is influenced by the two parameters. With 300 key definitions and 300 correspondences on the keys (which represents a complex case), the time for the join identification is about 15 seconds.

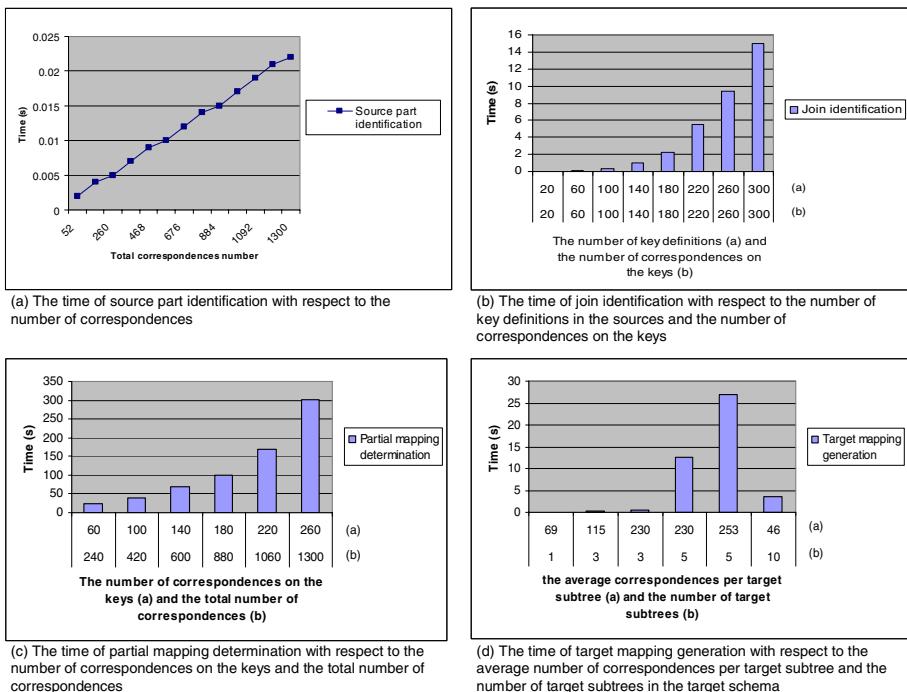


Fig. 10. Evaluating the time for the different steps of mapping generation

The time required for the determination of partial mappings for a given target subtree depends on the size of the corresponding join graph. Figure 10 (c) shows the time for partial mapping determination with respect to both the total number of correspondences and the number of correspondences for the keys. We have successively increased the values of these two parameters from 60 correspondences for the keys and 240 total correspondences to 260 correspondences for the keys and 1300 total correspondences. The other parameters of the scenarios used in this experiment are the same as the ABC2 scenario. We can see in the graph that a scenario having 880 correspondences among which 180 correspondences involving source keys takes about 100 seconds for the partial mapping determination.

The target mapping generation depends on the number of partial mappings and the structure of the target schema, that is, the number of parent-child relations between the target subtrees. Figure 10 (d) shows the time required for this task with respect to the number of the target subtrees and the average correspondences per subtree. We have increased both parameters using the same sources as for the ABC2 scenario. For example, in the case of 5 target subtrees and 230 correspondences per subtree, it takes about 12 seconds for generating the target mappings; note that this case is a complex one, since the scenario contains 50 sources, 1300 correspondences and 58 key definitions. The complexity of this process is exponential with respect to the number of partial mappings. It is possible to reduce this complexity using some quality criteria (for example, selecting the partial mappings that use the sources having a high confidence factor) or some heuristics (for example, selecting the partial mappings using a high number of sources).

7 Related Works

Several approaches [1][6][10] have been proposed to generate mappings when the target and the source schemas are expressed using the relational model. The approach presented in [1][6] generates a set of mappings from a set of source schemas using linguistic correspondences between target attributes and source attributes expressing that these elements represent the same concept. The work presented in this paper is inspired by this approach and also uses correspondences to define the mappings.

The approach presented in [10] generates a set of mappings from one source schema using a set of pre-defined value correspondences which specify how a target attribute is generated from one or more source attributes. In our work we also assume that correspondences are provided but we consider several source schemas.

Unlike the previously presented approaches ([1][6][10]), where the schemas are relational ones, we consider that either the target schema or the data sources are described in XML schema. In the case of XML sources, the complexity of mapping generation increases: we have to find instances for nodes of the tree representing the target schema, but also to preserve its structure.

An approach is proposed in [11] for generating mappings from one source schema to a target schema when these schemas are in XML Schema. In [14], a query rewriting algorithm which uses these mappings is proposed for integrating data sources. In our approach, the mappings are defined for a set of data sources; the mappings gener-

ated in our approach express the way instances of different schemas are combined to form instances of the target schema.

Other approaches have been proposed [2], [13], and [15] to generate mappings from several source schemas. These approaches comprise two steps: (i) the definition of rules to restructure each source schema according to the structure of the target schema; (ii) and the generation of mappings from these restructured schemas. In these approaches, source schemas must be restructurable with respect to the target schema in order to use them for mapping definition. In our approach, we do not impose such constraint, because some mapping may exist even if the restructuring of a source schema is not possible.

8 Conclusion

In this paper, we have presented algorithms for automatically generating mappings; we have implemented a system to support this task and presented some experimental results. This system produces a set of mappings for a target schema considering a set of source schemas and a set of correspondences; each target mapping has a different semantics.

Since the result of our system is a set of target mappings, one interesting perspective is to take advantage of these multiple semantics; the system should be able to select the mapping that most fits the needs of a specific user, using some preferences or some quality criteria. To achieve this goal, our system is already integrated in a platform that also provides tools for evaluating the quality of mappings considering user preferences [9]. Another perspective of our work is the maintenance of the mappings: if some changes occur in the data sources or in the target schema, some of the mappings may become inconsistent; the problem is therefore to detect the inconsistent mappings and to propagate the changes into the mapping definitions.

References

1. Bouzeghoub, M., Farias Lóscio, B., Kedad, Z., Salgado, A.-C.: Managing the evolution of mappings. Proc. of the 11th. Int. Conf. on Cooperative Information Systems (CoopIS'03), Catania, Italy (2003) 22-37
2. Claypool, K. T., Rundensteiner, E. A.: Gangam: A Transformation Modeling Framework. Proc. of Eighth Int. Conf. on Database Systems for Advanced Applications (DASFAA'03), Kyoto, Japan (2003) 47-54
3. Collet C., Belhajjame K., Bernot G., Bruno G., Bobineau C., Finance B., Jouanot F., Kedad Z., Laurent D., Vargas-Solar G., Tahí F., Vu T.-T., Xue X.: Towards a target system framework for transparent access to largely distributed sources. Proc of the Int. Conf. on Semantics of a Networked World Semantics for Grid Databases (IC-SNW'04), Paris, France (2004) 65-78
4. Dhamankar, R., Lee, Y., Doan, A., Halevy, A. Y., Domingos, P.: iMAP: Discovering Complex Mappings between Database Schemas. Proc. of Int. Conf. ACM SIGMOD (SIGMOD'04), Paris, France (2004) 383-394

5. Do, H.H., Rahm, E.: COMA - A System for Flexible Combination of Schema Matching Approaches. Proc. of the 28th Int. Conf. on Very Large Data Bases (VLDB'02), Hong Kong, China (2002) 610-621
6. Kedad, Z.; Bouzeghoub, M.: Discovering View Expressions from a Multi-Source Information System. Proc. of the 4th. Int. Conf. on Cooperative Information Systems (CoopIS'99), Edinburgh, Scotland (1999) 57-68
7. Kedad, Z., Xue, X.: Mapping Generation for XML Data Sources: a General Framework. Proc. of the Int. Workshop on Challenges in Web Information Retrieval and Integration (WIRI'05), in conjunction with the 21st Int. Conf. on Data Engineering (ICDE'05), Tokyo, Japan (2005)
8. Kostadinov, D., Peralta, V., Soukane, A., Xue, X. (Demonstration) : Système adaptif à l'aide de la génération de requêtes de médiation. Proc. of 20th Conf. of Bases de données avancées (BDA'04) ,Montpellier, France (2004) 351-355
9. Kostadinov, D., Peralta, V., Soukane, A., Xue, X.: Intégration de données hétérogènes basée sur la qualité. Proc. of INFORSID 2005 (Inforsid'05), Grenoble, France (2005) 471-486
10. Miller, R.J., Haas, L. M., Hernández, M. A.: Schema Mapping as Query Discovery. Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB'00), Cairo, Egypt (2000) 77-88
11. Popa L., Velegrakis Y., Miller R.J., Hernandez M.A., Fagin R.: Translating web data. Proc. of the 28th Int. Conf. on Very Large Data Bases (VLDB'02), Hong Kong, China (2002) 598-609
12. E. Rahm, P. A. Bernstein, A survey of approaches to automatic schema matching. Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB'01), Roma, Italy (2001) 334-350
13. Yang, X., Lee, M. L., Ling, T. W.: Resolving structural conflicts in the integration of XML schemas: a semantic approach. Proc. of 22nd Int. Conf. on Conceptual Modeling (ER'03), Chicago (2003) 520-533
14. Yu, C., Popa, L.: Constraint-based XML query rewriting for data integration. Proc. of Int. Conf. ACM SIGMOD (SIGMOD'04), Paris, France(2004) 371-382
15. Zamboulis, L., Poulovassilis, A.: XML data integration by Graph Restructuring. Proc. of the 21st Annual British National Conf. on Databases (BNCOD21), Edinburgh (2004) 57-71

Colored Petri Nets to Verify Extended Event-Driven Process Chains

Kees van Hee, Olivia Oanea, and Natalia Sidorova

Department of Mathematics and Computer Science,
Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
`{k.m.v.hee, o.i.oanea, n.sidorova}@tue.nl`

Abstract. Business processes are becoming more and more complex and at the same time their correctness is becoming a critical issue: The costs of errors in business information systems are growing due to the growing scale of their application and the growing degree of automation. In this paper we consider *Extended Event-driven Process Chains (eEPCs)*, a language which is widely used for modeling business processes, documenting industrial reference models and designing workflows. We describe how to translate eEPCs into timed colored Petri nets in order to verify processes given by eEPCs with the CPN Tools.

Keywords: Extended EPCs, semantics, verification, colored Petri nets.

1 Introduction

Event-driven Process Chains (EPC) [15,17] is a popular language for modeling business processes, documenting industrial reference models and designing workflows. EPCs describe the flow of control of business processes as a chain of functions, events, and logical connectors. Functions represent activities in a business process. An event expresses a precondition (trigger) for a function or a postcondition that signals the termination of a function. Logical connectors **and**, **or**, and **xor** are used according to their names to build the control flow of a process in a natural way.

EPCs extended with data, resources, time and probabilities, called *extended EPCs (eEPCs)* [17], are intensively used in commercial tools like Architecture of Integrated Information Systems (ARIS) [15] and SAP R/3 [11]. These tools support modeling and simulation of organizational processes with eEPCs, and they are widely used in such branches of industry and consultancy as banks, insurance companies, transportation. The complexity of business processes in these branches is growing throughout the years. Due to informatisation, which concerns all aspects of organizational activities, less and less manual work is involved into the supervision of business processes. This accelerates processes significantly, but also puts higher requirements to the correctness of process specifications, since an error in a process design would demonstrate itself in an automated system too late, when it would already cause a snowball effect.

We choose to use an available tool for modeling, simulation and analysis of the constructed model, e.g. model checking which covers the whole system behavior, and we provide a translation from eEPCs to the input language of this tool. Petri nets are appropriate for modeling EPCs since all EPC elements can be translated to places and transitions of Petri nets in a natural way (see e.g. [1,7,13]). Extended EPCs have such additional features as data, time and probabilities. Therefore timed colored Petri nets (TCPNs) [10] are a natural choice for modeling eEPCs. CPN Tools [3] provides modeling, simulation, and model checking options for TCPNs and thus satisfies our requirements.

In this paper, we provide a formal definition for eEPCs and present their semantics in terms of a transition system. We provide a translation from eEPCs to TCPNs and describe how we can analyze the behavior of an eEPC with the CPN Tools. We conclude by comparing our method to other approaches for formalizing the syntax and the semantics of EPCs.

The rest of the paper is organized as follows. In Section 2 we describe the syntax of eEPCs. In Section 3 we provide the semantics of eEPCs as used in practice. Section 4 gives a translation from eEPCs to timed colored Petri nets and discusses some verification issues. In Section 5 we give an account of related work and discuss some future work.

2 Syntax of Extended Event-Driven Process Chains

In this section, we give a brief description of the syntax of eEPCs taking into account requirements given in [15] as well as the ones imposed by practice. We use ARIS [6,9,15,17] as a reference point of our study. However, this approach can be applied to other tools supporting eEPCs, since they are based on the same concepts.

ARIS offers a conceptual framework for describing companies, their organizational structure, processes and resources (material as well as human). In addition to process modelling, ARIS offers the possibility to analyze process performance based on simulation. In order to structure process modelling and to show different angles of an organization, ARIS distinguishes five main views:

Data view uses the entity-relationship models (ERM) to design data models: entities (e.g. data objects of the environment that are processed by the system), their attributes and relationships between entities;

Function view describes functions as tasks performed on objects to support different company goals; it includes descriptions of procedures, processes, subfunctions and elementary functions;

Organization view models the relations between company units and the classification of these units in the organizational hierarchy;

Product/service view describes the products and services produced by the company as a result of human act or technical procedures;

Control view integrates the previously mentioned views and defines the dynamic, behavioral aspects. The control flow of a process is described with

an EPC extended with the description of the resources and data involved in the process, and timed and probabilistic aspects of the behavior.

The control view is essential for process verification, so we concentrate our study on this view. In what follows, we define generic EPCs and extend them to eEPCs as they are presented in the control view of ARIS.

2.1 Syntax of EPCs

First we give some basic definitions from algebra and the graph theory we need here.

- Let S be a set. $|S|$ denotes the number of elements in S . \mathbb{B} denotes the boolean set, \mathbb{N} the set of natural numbers, \mathbb{Z} the set of integers, \mathbb{R} the set of real numbers and \mathbb{R}^+ the set of positive real numbers.
- A multiset (bag) over S is a mapping $m: S \rightarrow \mathbb{N}$. The set of all multisets over S is \mathbb{N}^S . We use $+$ and $-$ for the sum and the difference of two multisets and $=, <, >, \leq, \geq$ for comparisons of multisets. \emptyset denotes the empty multiset and \in denotes element inclusion. We write $m = 2^a$ for a multiset m with $m(a) = 2$ and $m(x) = 0$ for any $x \in S - \{a\}$.
- Let $R \subseteq S \times S$ be a binary relation over a set S . R^{-1} denotes the converse relation of R , R^+ denotes the transitive closure of R , R^* denotes the transitive reflexive closure of R and $(R \cup R^{-1})^*$ is the symmetric, reflexive and transitive closure of R .
- A *directed graph* is a tuple $G = (N, A)$, where N is a set of nodes and $A \subseteq N \times N$ is a set of arcs. Every arc $a \in A$ is a pair $(n_1, n_2) \in N \times N$ consisting of the input node n_1 and the output node n_2 .
- A *path* σ of length m in a graph $G = (N, A)$ is a finite sequence of nodes $\sigma = n_0 n_1 \dots n_m$ ($n_i \in N$ for $i = 0, \dots, m$) such that each pair $(v_j, v_{j+1}) \in A$ ($j = 0, \dots, m - 1$) is an arc. We denote the length of a path by $|\sigma| (= m)$. σ is an empty path if $|\sigma| = 0$. We denote the set of all finite, possibly empty, paths by N^* and the set of finite non-empty paths by N^+ . A path σ is a prefix of a path γ if there is a path σ' with $\gamma = \sigma\sigma'$.
- A graph $G = (N, A)$ is *weakly connected* if for every two nodes $n_1, n_2 \in N$, $(n_1, n_2) \in (A \cup A^{-1})^*$.
- We denote the set of *output nodes* of $n \in N$ as n^\bullet , i.e. $n^\bullet = \{n' | (n, n') \in A\}$. Similarly, $\bullet n = \{n' | (n', n) \in A\}$ is the set of *input nodes* of $n \in N$. Given a set of nodes $X \subseteq N$, we define $\bullet X = \bigcup_{n \in X} \bullet n$ and $X^\bullet = \bigcup_{n \in X} n^\bullet$.
- We denote the set of *ingoing arcs* of a node $n \in N$ as A_n^{in} , i.e. $A_n^{in} = \{(n, x) | x \in n^\bullet\}$ and the set of *outgoing arcs* of n as A_n^{out} , i.e. $A_n^{out} = \{(x, n) | x \in n^\bullet\}$. In case A_n^{in} is singleton, a_n^{in} denotes the ingoing arc of the node n . Similarly, if A_n^{out} is singleton, a_n^{out} denotes the outgoing arc of the node n .

Now we can give a definition of a generic EPC.

Definition 1 (EPCs). *An event-driven process chain (EPC) is defined by a weakly connected directed graph $G = (N, A)$ that satisfies the following properties:*

1. The set N of nodes is the union of three pairwise disjoint sets E , F and C , where
 - E is the set of events. $E = E_s \cup E_f \cup E_i$, where E_s , E_f and E_i are pairwise disjoint sets of start events, final events, and internal events respectively, with $|E_s| \geq 1$ and $|E_f| \geq 1$;
 - $F \neq \emptyset$ is a set of functions;
 - C is a set of connectors of types **xor**, **or**, **and**, i.e. $C = C_{\text{xor}} \cup C_{\text{or}} \cup C_{\text{and}}$, where C_{xor} , C_{or} and C_{and} are disjoint sets. Furthermore, each of these sets is partitioned into two sets representing split and join connectors: $C_{\text{xor}} = C_{\text{xs}} \cup C_{\text{xj}}$, $C_{\text{or}} = C_{\text{os}} \cup C_{\text{oj}}$ and $C_{\text{and}} = C_{\text{as}} \cup C_{\text{aj}}$, and C_s stands for set of split connectors, and C_j stands for the set of join connectors;
2. Every element from the set A of arcs connects two different nodes. Moreover,
 - $\bullet e_s = \emptyset$ and $e_f^\bullet = \emptyset$, for each $e_s \in E_s$ and $e_f \in E_f$;
 - $|n^\bullet| = 1$ for each $n \in F \cup E_i \cup E_s$, and $|\bullet n| = 1$ for each $n \in F \cup E_i \cup E_f$;
 - each split connector $c \in C_s$ satisfies $|\bullet c| = 1$ and $|c^\bullet| > 1$; similarly each join connector $c \in C_j$ satisfies $|\bullet c| > 1$ and $|c^\bullet| = 1$;
3. Each node is on a path from a start event to a final event, i.e. for any $n \in N$, there is a path σ from some $e_s \in E_s$ to some $e_f \in E_f$, such that $n \in \sigma$;
4. Functions and events alternate along the control flow, i.e. each path starting in an event $e \in E - E_f$ and ending in an event $e_f \in E_f$ has a prefix of the form $e\sigma_c f$, where $\sigma_c \in C^*$ and $f \in F$. Similarly, for each path σ starting in a function $f \in F$ and ending in an event $e_f \in E_f$ there is a prefix $f\sigma_c e$, where $\sigma_c \in C^*$ and $e \in E - E_s$;
5. Events do not precede the **xor** and the **or** split, i.e. $\forall c \in C_{\text{xs}} \cup C_{\text{os}}: \bullet c \cap E = \emptyset$;
6. There is no cycle that consists of connectors only, i.e. for any path $\sigma = v_0v_1\dots v_n \in C^+: n \geq 2: v_0 \neq v_n$.

2.2 Syntax of Extended EPCs

The control view of an eEPC has an EPC as a skeleton. Data attributes, resources, time and probabilities are linked to different EPC elements to form an extended EPC.

Functions represent activities that may take time, may require access to diverse resources and may perform operations on some data or resources.

Functions that perform operations on data attributes are annotated with expressions denoting the operation performed (see for example Figure 1(a)). Personnel, material or information resources can be used to execute functions. We call these objects capacity resources, since they are characterized by minimum and maximum capacities to run the process. Functions are annotated with a nonnegative integer or real constant denoting the number of resources required, produced or consumed. In Figure 1(b), function *finish products* produces 1000 items of resource *Item 1* with the capacity domain [100, 5000] and consumes 500 items of resource *Item 2*.

Furthermore, functions can be either *timed*, i.e. they may have a duration, or *immediate*, i.e. they take zero time. The duration of each timed function is described by a probability distribution.

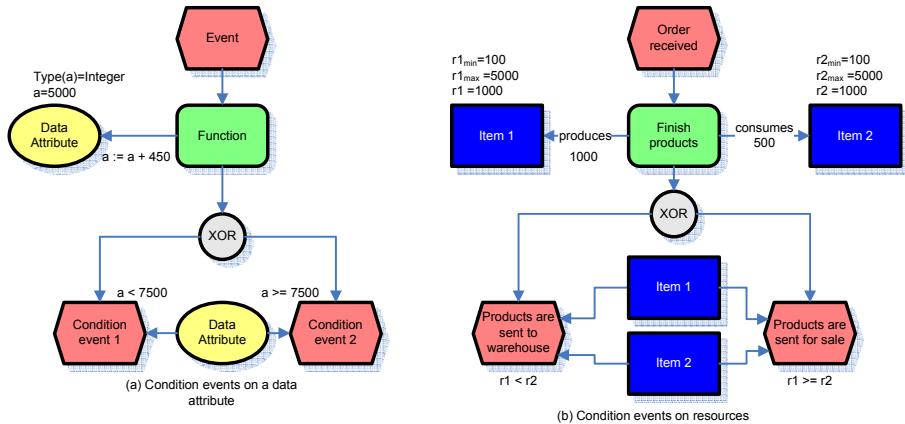


Fig. 1. Condition events on data attributes and resources

Events define either conditions on data attributes or resource capacities, or triggers from elements outside the process.

Processes are instantiated at **start events**. Start events may be grouped in **start event sets**¹ that contain events which are synchronized, i.e. a process is started at the same time at all the events of the respective set. A probability distribution is assigned to each start event set, denoting the frequency with which process instances are created for the events in the respective set.

Conditions (boolean expressions) on data attributes or on resources determine the terms of the respective event. An event that follows an **or** split or an **xor** split connector and is determined by a condition is called a **condition event**. Condition events may have **attributes** or **capacity resources** connected to them and conditions are specified as:

- conditions on one operand that have a constant value of the same type as the attribute or the capacity value of a resource as comparison criterion. Figure 1(a) shows two condition events annotated with boolean expressions on a data attribute;
- conditions on two operands that compare two attribute values or the capacity values of two resources. In Figure 1(b), the condition events *products are sent to warehouse* and *products are sent for sale* are annotated with the boolean expressions $r1 < r2$ and $r1 \geq r2$ on the resources *Item 1* and *Item 2*.

The rest of events are used to model triggers from outside the process and they have a probability value assigned. This value is used during the simulation to determine whether the execution stops or continues at the respective event. Probability values for events following **and** split connectors are 1 since the execution cannot stop at events following such a connector. The sum of probability

¹ In ARIS, event diagrams are used to represent start event sets.

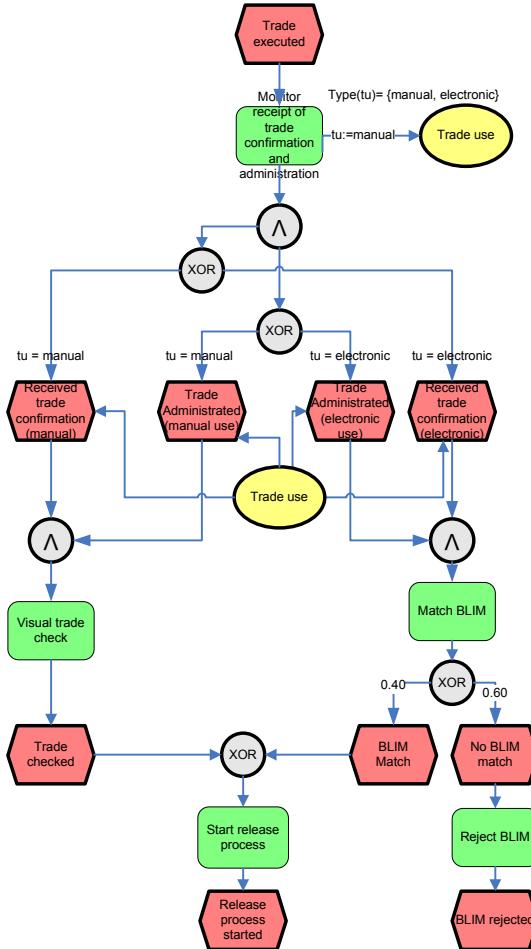
values for events following **xor** split connectors is exactly 1 as the execution can continue only on one outgoing branch. Furthermore, the sum of probability values for events following **or** split connectors is greater than 1 as the execution can continue on one or more outgoing branches.

Or join connectors may also contain some timeout information, called *synchronization timeout*.

We give a formal definition of eEPCs as they are used in ARIS Toolset.

Definition 2 (eEPC). *An extended event-driven process chain (eEPC) is a tuple $G_e = (G, \mathcal{A}, \mathcal{R}, \text{Type}, \text{Expr}, \text{PDF}, \text{Pr})$, where*

- G is an underlying EPC.
- \mathcal{A} is a set of data attributes. We write $\mathcal{A} = \bigcup_{f \in F} \mathcal{A}_f$ for a partition of the set of data attributes w.r.t. the function performing operations on them.
- \mathcal{R} is a set of capacity resources. We partition the set of resources according to the function performing operations on them: $\mathcal{R} = \bigcup_{f \in F} \mathcal{R}_f$ such that these sets are not disjoint (several functions can perform operations on the same resource). Moreover, we consider a partition of the set of resources used by a function f into used, produced and consumed resources, i.e. $\mathcal{R}_f = \mathcal{R}_f^u \cup \mathcal{R}_f^p \cup \mathcal{R}_f^c$ such that these sets are disjoint (a function can perform just one type of operation on a resource).
- **Type** maps each attribute to one of the types **Text**, **Enum**, \mathbb{B} , \mathbb{Z} , or \mathbb{R} and each capacity resource r to a real or integer subtype $[r_{\min}, r_{\max}]$, where r_{\min} and r_{\max} are the minimum and maximum capacities of resource r .
- **Expr** = $\text{Expr}^b \cup \bigcup_{x \in \mathcal{R} \cup \mathcal{A}} \text{Expr}_x$ maps condition events and functions into expressions on the attributes or capacity resources linked to them as follows:
 - E_c denotes the set of **condition events**, i.e. events following **or** and **xor** split connectors that have conditions on data attributes or resources. Every condition event $e \in E_c$ is mapped to a boolean expression $\text{Expr}^b(e)$ of the form $v_1 \mathbf{x} v_2$ or $v_1 \mathbf{x} c$, where v_1, v_2 are either attributes or resource capacities, c is a constant so that $\text{Type}(v_1) = \text{Type}(v_2) = \text{Type}(c)$ and \mathbf{x} is the comparison operator compatible with the types used.
 - every function f performing an operation on an attribute a is mapped to an expression on a , namely $\text{Expr}_a(f)$, having the form $a := c$, where c is a constant value with $\text{Type}(a) = \text{Type}(c)$, or $a := a \mathbf{x} c$, with $\text{Type}(a) = \mathbb{Z}$ or $\text{Type}(a) = \mathbb{R}$, constant c with $\text{Type}(a) = \text{Type}(c)$ and $\mathbf{x} \in \{+, -, *\}$.
 - $\text{Expr}_r(f)$ maps function f using, producing or consuming a resource $r \in \mathcal{R}_f$ to constant $c_r^f \in \text{Type}(r)$, denoting the quantity of resources used, consumed or produced.
- F_t denotes the timed functions, and C_s denotes the set of **or** join connectors with synchronization timeout. We consider the set of start events E_s partitioned into start event sets, i.e. $\bigcup_{d \in I_s} E_s^d$, for some index set I_s .

**Fig. 2.** Trade matching eEPC

$PDF = \bigcup_{k \in (F_t \cup C_s \cup I_s)} pdf_k$ denotes a family of continuous or discrete probability distributions² for the duration of timed functions, for the synchronization timeouts of synchronized or join connectors, and for the delays of start event sets.

- **Pr:** $E - E_s - E_c \rightarrow [0, 1]$ which maps events to their probability values such that:
 - $\sum_{e \in E^c} \mathbf{Pr}(e) = 1$, for each set of events E^c following an **xor** split connector $c \in C_{\mathbf{xs}}$ that have probability values;

² In this paper, a probability distribution $pdf \in PDF$ refers to a probability distribution function (we do not mention the word function in order to avoid confusion with functions as nodes of eEPCs).

- $\sum_{e \in E^c} \mathbf{Pr}(e) \geq 1$, for each set of events E^c that have probability values and follow an **or** split connector $c \in C_{\text{os}}$;
- $\mathbf{Pr}(e) = 1$ for each $e \in C_{\text{as}}$.

Figure 2 shows an eEPC modeling a part of a trade matching process taking place in a company. The process checks the timely receipt of a confirmation, administrates the trade internally and matches the confirmation against the internal data before the release process can be started.

The process starts with the event *trade executed (deal made)* that triggers the function *monitor receipt of trade confirmation and administration*. This results in a change of the data attribute *trade use*. The execution is then split into two parallel threads, which is modeled by an **and** split. The left thread models the check whether the confirmation of the trade has been received electronically or manually by means of an **xor** split and two condition events: *Received trade confirmation (manual)* and *Received trade confirmation (electronic)* that are linked to a data attribute *Trade use*. The second thread models the check whether the trade is administered for manual or electronic use by means of conditions on the attribute *trade use*.

The two **and** join connectors make sure that the matching process continues either manually or electronically. The visual (manual) check is performed by the function *visual trade check* and results in the event *trade checked*. The result of the electronic matching of the internal information with BLIM messages has 40% probability to succeed. In case the trade has been matched either manually or electronically, which is modeled by an **xor** join, the process is released by *start release process*. If the data registered internally does not match the information contained in the BLIM message, the message is rejected.

3 Semantics of eEPCs

We introduce first the notions of a *process folder* and a *state* of an eEPC necessary to define the semantics of eEPCs.

A **process folder** is an object that resides at a node (function or start event) or at an arc. Furthermore, it carries a folder number and a timestamp denoting the value of the timer associated to the folder. Timestamps are either nonnegative numbers indicating the delay after which the folder may be used, or \perp , denoting that the timer of the process folder is off and the folder can be used directly. A **state** of an eEPC is defined by a multiset of process folders together with a valuation of data attributes and capacity resources.

For the rest of the paper, we consider the discrete time domain $\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$ and discrete probability distributions for durations. We denote the domain of a discrete probability distribution $pdf \in PDF$ by $Dom(pdf) \subseteq \mathbb{N}$. The same approach can be applied for continuous time and continuous probability distributions. We consider process folder numbers to take values from \mathbb{N} . Formally:

Definition 3. Let $G_e = (G, \mathcal{A}, \mathcal{R}, \mathbf{Type}, \mathbf{Expr}, PDF, \mathbf{Pr})$ be an eEPC. A process folder is a tuple $p = (n, (i, t))$ where $n \in E_s \cup F \cup A$ is a start event,

a function or an arc, $i \in \mathbb{N}$ is a process folder number and $t \in \mathbb{N}_\perp$ is a timestamp. A state of G_e is a tuple $s = (m, \mathbf{Val})$, where m is a multiset of process folders, i.e. $m: ((F \cup A \cup E_s) \times (\mathbb{N} \times \mathbb{N})) \rightarrow \mathbb{N}$, and \mathbf{Val} is a valuation function that maps every resource $r \in \mathcal{R}$ into some value $\mathbf{Val}(r) \in \mathbf{Type}(r)$ and every data attribute $a \in \mathcal{A}$ to some value $\mathbf{Val}(a) \in \mathbf{Type}(a)$.

We denote the timestamp of a process folder p by p_t . We will say that a process folder has its timer *off* when $p_t = \perp$ and its timer *on* when $p_t \geq 0$. We call a process folder with the timer on *active* if it has the timestamp 0. If $p_t > 0$, the process folder is *waiting* to become active.

Every start event of a start event set has initially an active process folder with the index of the start event set as its folder number. The initial state is thus $s_0 = (m_0, \mathbf{Val}_0)$ and contains one active process folder on every start event so that for every start event set $(e_s, (i, 0)) \in m_0$ for all $e_s \in E_s^i$, and every resource and data attribute has an initial value according to the specification.

Probability distributions are used in eEPCs to model the behavior of the environment or to describe nondeterminism in the system (when decisions need to be made), and for performance analysis. Since all events that have $\mathbf{Pr}(e) > 0$ can occur and the errors in a model (eEPC) having probabilistic events can thus be detected on the model without probabilities, we do not take probabilities further into consideration, as they are irrelevant to our verification purposes. In order to express nondeterminism without probabilities, we extend the mapping **Expr** to non-condition events and set $\mathbf{Expr}^b(e) = \text{true}$, for every event $e \in (C_{\text{os}} \cup C_{\text{xs}})^\bullet - E_c$, and subsequently extend the set of *condition events* to all events following an **or** and **xor** split connector, i.e. $E_c = (C_{\text{os}} \cup C_{\text{xs}})^\bullet$.

Let *eval* be the *evaluation function* for expressions, such that:

- $\text{eval}(\mathbf{Expr}^b(e), \mathbf{Val}) \in \mathbb{B}$ is the evaluation function of the boolean expression of condition events $e \in E_c$ in the valuation \mathbf{Val} .
- $\text{eval}(\mathbf{Expr}_a(f), \mathbf{Val}) \in \mathbf{Type}(a)$ is the evaluation function of the expression $\mathbf{Expr}_a(f)$ on some data attribute $a \in \mathcal{A}_f$ in the valuation \mathbf{Val} that computes a new value for a from the right hand side expression of $\mathbf{Expr}_a(f)$.

For any $r \in \mathcal{R}$, we introduce a variable \bar{r} such that $\mathbf{Val}_0(r) = \mathbf{Val}_0(\bar{r})$ in order to keep track of resources that would be modified by several functions. We denote the set of variables newly introduced by $\overline{\mathcal{R}}$.

We describe the semantics of an eEPC by means of a transition relation between states as follows:

Definition 4. Let $G_e = (G, \mathcal{A}, \mathcal{R}, \mathbf{Type}, \mathbf{Expr}, \mathbf{PDF}, \mathbf{Pr})$ be an eEPC. The semantics of an eEPC is given by a transition system $TS = (\Sigma, s_0, \rightarrow)$, where Σ is the set of states of G_e , s_0 is the initial state, and $\rightarrow \subseteq \Sigma \times \Sigma$ is a transition relation described by the rules (a) – (j) below. Let $s = (m, \mathbf{Val})$ and $s' = (m', \mathbf{Val}')$ be two states in Σ .

- (a) **start event set rule** Let E_s^d be a start event set such that there is a process folder on each of its start events and all the folders have the same folder number and are active. Then a step can be taken that results in removing all the folders on the events of E_s^d and producing a process folder on each of the outgoing arcs of E_s^d which have the same folder number as the original process folders and their timers are set to off. Furthermore, a new process folder is generated on each event of E_s^d ; all these new process folders have the same newly generated folder number and the same timestamp which is drawn from the probability distribution of the start event set. Formally, if $(e, (i, 0)) \in m$ for all $e \in E_s^d$, $d \in I_s$ and $i \in \mathbb{N}$, then $s \rightarrow s'$, with $m' = m - \sum_{e \in E_s^d} (e, (i, 0)) + \sum_{e \in E_s^d} (a_e^{out}, (i, \perp)) + \sum_{e \in E_s^d} (e, (i + |I_s|, t))$, for some $t \in \text{Dom}(\text{pdf}_d)$.
- (b) **event rule** Let a_e^{in} be the incoming arc for an event e such that there is a process folder on a_e^{in} . Then, the process folder can be removed from a_e^{in} and placed on the outgoing arc of the event a_e^{out} . Formally, if $(a_e^{in}, (i, \perp)) \in m$ for some $i \in \mathbb{N}$ and $e \in E - E_s - E_c - E_f$, then $(m, \text{Val}) \rightarrow (m', \text{Val})$ with $m' = m + (a_e^{out}, (i, \perp)) - (a_e^{in}, (i, \perp))$.
- (c) **function rule** Let a_f^{in} be the incoming arc of a function f such that there is a process folder on a_f^{in} . The function rule consists of two steps:
- if the resources may be used, consumed or produced, the process folder is removed from the incoming arc of the function, and a new process folder having the same folder number as the original folder and a timestamp from the time distribution interval of the function is placed on the function, and resources are consumed.
- Formally, if $(a_f^{in}, (i, \perp)) \in m$, for some $f \in F$ and $i \in \mathbb{N}$, $\text{Val}(r) - \text{Expr}_r(f) \leq r_{\min}$ for all $r \in \mathcal{R}_f^u \cup \mathcal{R}_f^c$, $\text{Val}(\bar{r}) + \text{Expr}_r(f) \leq r_{\max}$ for all $r \in \mathcal{R}_f^p$, then $s \rightarrow s'$, where $s' = (m', \text{Val}')$, $m' = m - (a_f^{in}, (i, \perp)) + (f, (i, t))$, where $t = 0$ if $t \in F - F_t$ and $t \in \text{Dom}(\text{pdf}_f)$ if $t \in F_t$, $\text{Val}'(r) = \text{Val}(r) - \text{Expr}_r(f)$ for all $r \in \mathcal{R}_f^c \cup \mathcal{R}_f^u$, $\text{Val}'(\bar{r}) = \text{Val}(\bar{r}) - \text{Expr}_r(f)$ for all $r \in \mathcal{R}_f^c$, $\text{Val}'(\bar{r}) = \text{Val}(\bar{r}) + \text{Expr}_r(f)$ for all $r \in \mathcal{R}_f^p$ and $\text{Val}'(x) = \text{Val}(x)$ for all $x \in (\mathcal{R} \cup \bar{\mathcal{R}} \cup \mathcal{A}) - (\mathcal{R}_f^c \cup \mathcal{R}_f^u \cup \bar{\mathcal{R}}_f^c \cup \bar{\mathcal{R}}_f^p)$.
- Let f be a function such that there is an active process folder on it. Then, the active process folder is removed from the function, and a new process folder is produced on the outgoing arc of the function; this new folder has the same folder number as the removed one and the timer off; attributes are evaluated and resources are produced or released. Formally, if $(f, (i, 0)) \in m$, for some $f \in F$ and $i \in \mathbb{N}$, then $s \rightarrow s'$, with $s' = (m', \text{Val}')$, $m' = m - (f, (i, 0)) + (a_f^{out}, (i, \perp))$, where $\text{Val}'(a) = \text{eval}(\text{Expr}_a(f), \text{Val})$ for all $a \in \mathcal{A}_f$, $\text{Val}'(r) = \text{Val}(r) + \text{Expr}_r(f)$ for all resources $r \in \mathcal{R}_f^p \cup \mathcal{R}_f^u$ produced or used, and $\text{Val}'(x) = \text{Val}(x)$ for all $x \in (\mathcal{R} \cup \bar{\mathcal{R}} \cup \mathcal{A}) - (\mathcal{A}_f \cup \mathcal{R}_f^p \cup \mathcal{R}_f^u)$.
- (d) **and split rule** Let a_c^{in} be the ingoing arc of an **and** split connector c such that there is a process folder on a_c^{in} . The rule results in removing the process folder from a_c^{in} and placing a process folder on each outgoing arc of the **and** split connector such that all new process folders have the same folder number

as the removed folder. Formally, if $(a_c^{in}, (i, \perp)) \in m$, for some $i \in \mathbb{N}$ and $c \in C_{\text{as}}$, then $s \rightarrow s'$ with $s' = (m, \text{Val})$ and $m' = m + \sum_{a' \in A_c^{out}} (a', (i, \perp)) - (a_c^{in}, (i, \perp))$.

- (e) **xor split rule** Let a_c^{in} be the incoming arc of an **xor** split connector c such that there is a process folder on a_c^{in} . Then the process folder is removed from the arc a_c^{in} and

- if the **xor** split leads to a non-final condition event whose boolean expression is evaluated to true, a process folder with the same number as the removed one is placed on the outgoing arc of the event;
- if there is an outgoing arc of the **xor** split leading to a final condition event whose boolean expression is evaluated to true or to another connector, then a process folder with the same number as the removed one is placed on the respective arc.

Formally, if $(a_c^{in}, (i, \perp)) \in m$, for some $i \in \mathbb{N}$ and $c \in C_{\text{xs}}$, then $s \rightarrow s'$, with $s' = (m', \text{Val})$ and $m' = m + (a', (i, \perp)) - (a_c^{in}, (i, \perp))$, where

- $a' = a_e^{out}$ if $\text{eval}(\text{Expr}^b(e), \text{Val}) = \text{true}$ for some event $e \in c^\bullet - E_f$, or
- $a' = (c, e)$, if $\text{eval}(\text{Expr}^b(e), \text{Val}) = \text{true}$ for some final event $e \in c^\bullet \cap E_f$, or
- $a' = (c, c')$ for some $c' \in C$.

- (f) **or split rule** Let a_c^{in} be the ingoing arc of an **or** split connector c such that there is a process folder on a_c^{in} . The rule results in removing this folder from a_c^{in} and placing a process folder with the same folder number on at least one of the outgoing arcs of the non-final condition events that have a true boolean expression or the outgoing arcs of the **or** split connector leading to final condition events with boolean expression evaluated to true or to connectors. Formally, if $(a_c^{in}, (i, \perp)) \in m$, for some $i \in \mathbb{N}$ and $c \in C_{\text{os}}$, then $s \rightarrow s'$, with $s' = (m', \text{Val})$ and $m' = m - (a_c^{in}, (i, \perp)) + \sum_{a \in A' \cup A''} (a, (i, \perp))$, where $A' \subseteq \{a_e^{out} | e \in c^\bullet \cap (E_c - E_f) \wedge \text{eval}(\text{Expr}^b(e), \text{Val}) = \text{true}\}$ and $A'' \subseteq \{(c, e) \in A_c^{out} | e \in c^\bullet \cap (E_c \cap E_f) \wedge \text{eval}(\text{Expr}^b(e), \text{Val}) = \text{true}\} \cup \{(c, c') \in A_c^{out} | c' \in C\}$ and $A \cap A'' \neq \emptyset$.

- (g) **and join rule** Let A_c^{in} be the set of all incoming arcs of an **and** join connector c . If all arcs of A_c^{in} have process folders with the same folder number, the rule can be applied resulting in the removal of these process folders from A_c^{in} , and the production of a process folder with the same process folder number as the original folders on the outgoing arc of the connector. Formally, if there is a $c \in C_{\text{aj}}$ such that $(a, (i, \perp)) \in m$, for all arcs $a \in A_c^{in}$ and some $i \in \mathbb{N}$, then $s \rightarrow s'$ with $s' = (m', \text{Val})$, and $m' = m + (a_c^{out}, (i, \perp)) - \sum_{a \in A_c^{in}} (a, (i, \perp))$.

- (h) **xor join rule** Let a be an incoming arc of an **xor** join connector such that there is a process folder on a . Then, the folder is removed from a and placed on the outgoing arc of the connector. Formally, if $(a, (i, \perp)) \in m$ for some $i \in \mathbb{N}$, $c \in C_{\text{xj}}$ and $a \in A_c^{in}$, then $s \rightarrow s'$ with $s' = (m', \text{Val})$, $m' = m + (a_c^{out}, (i, \perp)) - (a_c^{in}, (i, \perp))$.

(i) **or join rule** Let A' be the set of incoming arcs of an **or** join connector that have process folders with timers off and the same folder number. The rule consists of one or more steps (depending on whether the connector has a synchronization time or not):

- (**or join unsynchronized**) In case the **or** connector does not have a synchronization timeout, the rule results in removing all the folders with timers off and the same folder number on A' and producing a process folder with the same folder number as the original folders and the timer off on the outgoing arc of the connector. Formally, if there is a $c \in C_{\text{obj}} - C_s$ such that $A' = \{a \in A_c^{in} | (a, (i, \perp)) \in m\} \neq \emptyset$, for some $i \in \mathbb{N}$, then $s \rightarrow s'$ with $s' = (m', \text{Val})$ and $m' = m + (a_c^{out}, (i, \perp)) - \sum_{a' \in A'} (a', (i, \perp))$.
- (**or join synchronized waiting**) Let A'' be a set of incoming arcs of a synchronized **or** join connector that have waiting process folders on them with the same folder numbers as the folders on A' . The rule results in removing the process folders with timers off and the same folder number on A' and producing new process folders on A' having the same folder number as the removed ones and a timestamp that is either the synchronization time of the **or** connector in case A'' is empty or the timestamp of the folders in A'' if A'' is non-empty. Formally, let $A' = \{a = (x, c) \in A | x \in N \wedge (a, (i, \perp)) \in m\} \neq \emptyset$ and $A'' = \{a \in A_c^{in} | (a, (i, t'')) \in m \wedge t'' > 0\}$ for some $i \in \mathbb{N}$ and $c \in C_s$. If $A'' = \emptyset$ then let $t' = t$ be the timestamp of the process folders $p = (a, (i, t)) \in m$, where $a \in A''$, otherwise let $t \in \text{Dom}(pdff)$. Then, $s \rightarrow s'$ with $s' = (m', \text{Val})$ and $m' = m + \sum_{a' \in A'} (a', (i, t')) - \sum_{a' \in A'} (a', (i, \perp))$.
- (**or join synchronized firing**) Let A'' be the set of incoming arcs of an **or** connector that have an active process folder on them and all these folders on A'' have the same folder number. Then these process folders are removed from A'' and a new process folder is produced on the outgoing arc of the connector, so that it has the same folder number as the original folders and the timer off. Formally, if for some $c \in C_{\text{obj}} - C_s$ and $i \in \mathbb{N}$, $A'' = \{a \in A_c^{in} | (a, (i, 0)) \in m\} \neq \emptyset$, then $s \rightarrow s'$ with $s' = (m', \text{Val})$ and $m' = m + (a_c^{out}, (i, \perp)) - \sum_{a' \in A''} (a', (i, 0))$.

(j) **time step rule** This rule has the lowest priority, i.e. the rule is applied if no other rule can be applied. Time passage is applied when all process folders with timers on in a state are waiting (have a strictly positive timestamp) and results in decreasing the timestamp of all process folders of the state by the minimal timestamp of the folders with timers on. Formally, if $P' = \{p \in m | p_t > 0\} \neq \emptyset$ and $s_t = \min\{p_t | p \in P'\} > 0$ and there is no other state $s'' \neq s'$ such that $s \rightarrow s''$, then $s \rightarrow s'$, with $s' = (m', \text{Val})$, where $m' = m + \sum_{(x, (i, t)) \in P'} (x, (i, t - s_t)) - \sum_{p \in P'} p$.

Remark 5 (Uniqueness of newly generated folder numbers). Folder numbers generated at some start event set E_s^d ($d \in I_s$) have the form $d + k \cdot n$, where $n = |I_s|$ and $k \in \mathbb{N}$. Therefore, all folder numbers produced at E_s^d are equal modulo the

index number of the start event set, i.e. d . As a result, all process folder numbers that are generated are unique.

Remark 6 (Time progress). The time step rule decreases the timestamp of a waiting process folder until it becomes active (its timer expires). Note that all other steps have the same higher priority than the time progress and their semantics is interleaving. The time progress problem reduces to the situation where no other rule can be applied or to the situation where there is an infinite sequence of (timeless) rules that can be applied. We therefore assume that there is a finite number of process folder numbers such that the states contain a finite number of process folders with the same folder number.

4 Verification of eEPCs Using CPN Tools

To verify the correctness of eEPCs, we use the CPN Tools [3] which are based on colored Petri nets [10] as modeling and analysis language. (Timed) Colored Petri nets (TCPNs) combine the expressive power of classical PNs, which are suitable for modeling the logical behavior of a control flow, with the modeling of data and time by means of timed color sets and a global clock. A state of a TCPN is called a *marking* and represents the distribution of (timed) colored tokens on places. Tokens belonging to a timed color set have a timestamp and they can only be used if the value of the timestamp is less than the value of the global clock.

Let G_e be an arbitrary eEPC and $TS = (\Sigma, s_0, \rightarrow)$ the transition system describing the semantics of G_e . We denote the timed integer color corresponding to the set of process folders numbers by **PF**, and for each capacity resource and data attribute, we define a place having the corresponding untimed color type. The locations at which process folders can reside correspond to TCPN places having type **PF** and local steps in TCPNs are depicted by transitions. A step that can be taken in the eEPC corresponds to a transition firing, given preconditions and postconditions expressed as expressions on arcs or guards (boolean expressions) on transitions. The global clock (model time) advances the timestamps of all timed tokens with the smallest amount of time needed to make at least one token active, which corresponds to the time step rule in eEPCs in which timers decrease their values. Token delays are positioned on transitions or on outgoing arcs of transitions.

4.1 Transformation of eEPCs into TCPNs

For mapping of eEPCs into TCPNs, we first identify generic eEPC patterns and provide their translation into TCPN patterns, and then we show how the obtained TCPN patterns can be fused together to form a TCPN.

We define eEPC patterns taking into account the rules of the semantics given in Section 3 (except for the time step rule), covering all patterns that would allow us to build an arbitrary eEPC. An eEPC pattern consists of incoming and outgoing arc(s) and other elements necessary for the rule to occur.

Figures 3(a-i) and 4 show instances of these patterns having the incoming (outgoing) arcs dotted and their corresponding TCPN pattern. In what follows we describe these transformations in more detail.

Start event set pattern. Let E_s^d be a start event set for some $d \in I_s$ and let $n = |E_s^d|$. The corresponding TCPN pattern is represented by a place which is initially marked with a token $d@0$ (with timestamp 0) and where a newly generated token with a delay is deposited; a transition that generates a new token, adds a delay to its timestamp and puts the token from the former place on the places corresponding start events of the start event set. Figure 3(a) shows the eEPC pattern and the TCPN pattern for a start event set with two start events.

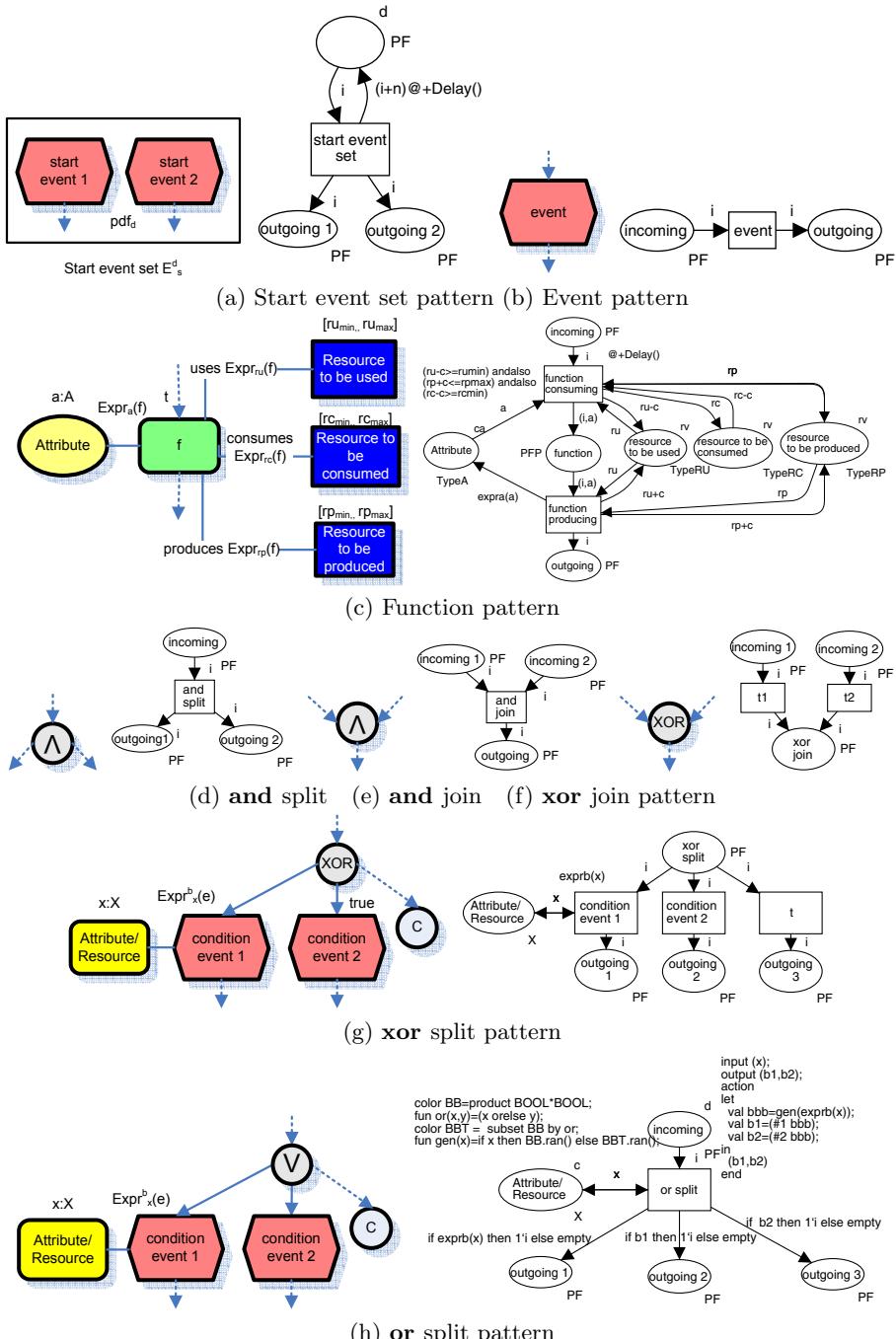
Event pattern. Let $e \in E - E_s - E_f - E_c$ be an event. The corresponding TCPN pattern is shown in Figure 3(b). Note that final events are not translated.

Function pattern. Let f be a (timed) function connected to the set of resources \mathcal{R}_f and to the set of data attributes \mathcal{A}_f . The corresponding TCPN pattern has two transitions and an intermediate place depicting the two steps of the function rule. Figure 3(c) shows the pattern and its translation operating on an attribute, a resource that is used, a resource that is produced and a resource that is consumed. The operations on the resources and attributes are described on the arc inscriptions.

and split, and join, and xor join patterns. These patterns are parameterized by the number of the outgoing and incoming arcs, respectively. Figure 3(d-f) shows the respective patterns with two incoming, respectively outgoing arcs;

xor split pattern. Let c be an **or** split connector. The TCPN pattern consists of a place with outgoing transitions. The boolean expressions on the condition events of c (on resources or attributes) become guards for the transitions. Figure 3(g) shows an **or** split connector having a condition event linked to an attribute or resource, a condition event with condition *true* and a connector on the outgoing arcs and the corresponding TCPN pattern.

or split pattern. Let c be an **or** split connector. The TCPN pattern contains a transition and a non-deterministic choice is implemented in the code segment of the transition by means of generation of boolean variables for each outgoing arc of the transition corresponding to outgoing arcs leading to non-conditional events. Figure 3(h) shows an instance of the pattern and its translation. In case at least one of the boolean expressions corresponding to conditions on attributes or resources is true, a boolean variable generated for each arc that does not lead to a condition event with conditions on attributes or resources. In case all the boolean expressions on attributes or resources are false or there are no condition events on data attributes or resources, the boolean variables generated for the rest of the arcs must have at least one *true* value. The boolean expressions on the condition events and the boolean values generated become conditions in the arc inscriptions.

**Fig. 3.** Translation of the eEPC patterns into TCPN patterns

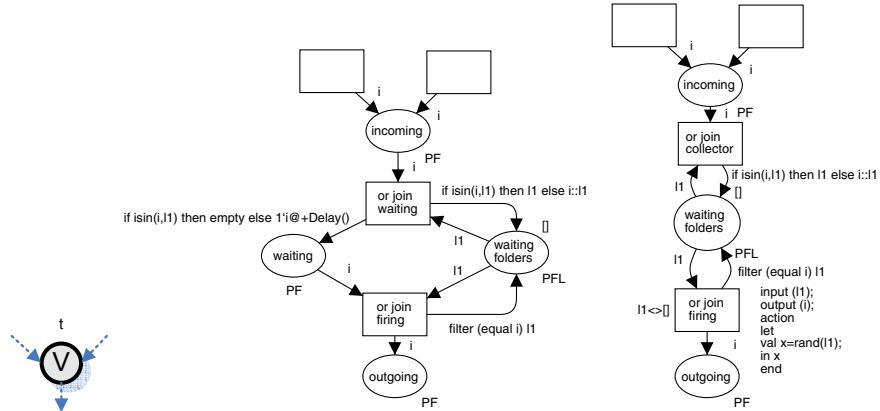


Fig. 4. or join pattern (synchronized and unsynchronized)

or join pattern. Figure 4 shows the pattern with two incoming arcs. In the TCPN pattern, place *incoming* collects all the tokens on the incoming arcs. The *or join* waiting transition decides whether the token will receive a delay by checking if the folder number is already in the list of waiting folders (in the place *waiting folders*). The *or join firing* transition becomes enabled, then the timestamp of the tokens in place *waiting* expires and the firing removes the token from the list of waiting tokens present in the place *waiting folders*. Note that in the untimed version the place *waiting* is eliminated.

Two eEPC patterns are called *adjacent* if an outgoing arc of a pattern coincides with an ingoing arc of another pattern. Two TCPN patterns are called *adjacent* if the corresponding eEPC patterns are adjacent. Adjacent TCPN patterns are fused, i.e. for every two adjacent patterns, if they have the same input and output nodes (e.g. both are either transitions or places), then the two nodes are fused; otherwise, a directed arc is added between them.

4.2 Verification

A TCPN obtained using the translation procedure described in the previous section can be simulated and analyzed by the CPN Tools [3], using state-space analysis (which is basically an exhaustive search through all possible states of the model).

The first check on eEPCs to be performed is whether the semantics of diverse connectors is respected. For **xor** join connectors, this coincides with checking whether there are reachable markings having two tokens in the places corresponding to **xor** join connectors. If there are, we can conclude that the eEPC model is not correct and we can provide a simulation of the TCPN that leads to this error.

If an **or** split has condition events on all its outgoing arcs, at least one of the conditions should be evaluated to *true* at every reachable marking. The violation of this requirement can be easily checked by finding a marking with at least one token on the place corresponding to the incoming arc of the **or** split such that all boolean expressions on resources or data attributes are *false*.

CPN Tools can provide a report on the state space of the constructed TCPN that includes information about dead markings (marking at which no transition is enabled). In case the only dead markings of the TCPN are markings having tokens on the places corresponding to incoming arcs of final events, we can conclude that the original eEPC *terminates properly*. Dead markings can also provide information about deadlocks in the eEPC, e.g. functions that cannot execute due to non-availability of resources or non-synchronization. Furthermore, the CPN Tools can verify properties of the model which are defined as temporal logic formulas in ASK-CTL [4].

5 Related and Future Work

Related work. There are different approaches to the formalization of the syntax and semantics of EPCs.

One approach is to use Petri nets to specify their semantics. An EPC is translated into a PN using a set of transformation rules. The semantics of EPCs is defined as the semantics of resulting Petri nets. Van der Aalst [1] and Dehnert [7] use a subclass of PNs — workflow nets that is suitable to describe EPCs and use specific verification methods developed for PNs in order to verify EPCs. In [1], an EPC is considered to be correct if and only if the workflow obtained as the translation of an EPC is sound. Langner, Schneider and Wehler [13] use a transformation into boolean nets which are colored Petri nets with a single color of type boolean and formulas from the propositional logic as guards. The correctness criterion is the well-formedness of the corresponding boolean net, which is too strict for some practical applications.

Another approach is to consider the transition systems-based semantics. In [16], [2] and [12], the dynamic behavior of an EPC is defined in terms of transition systems. In [2] and [12], the state of an EPC is defined as a mapping of the set of arcs to $\{0, 1\}$ and is represented by the presence or absence of process folders on the arcs of the EPC. Moreover, [2] proposes a non-local semantics of the **xor** and **or** join connector that refers to checking certain conditions that depend on the overall behavior of the EPC. An **xor** join has to wait for a folder on one of its input arcs in order to propagate it to its output arc. However, if a process folder is present or could arrive at some other input arc, the **xor** join should not propagate the folder. For an **or** join, the propagation of a process folder from its input arcs is delayed as long as a process folder could possibly arrive at one of the other input arcs. Computing the transition relation of an EPC has been implemented and tested in [5] using symbolic model checking.

In our paper we consider the semantics of extended event-driven process chains, i.e. EPCs extended with data, resources, and time as it is specified in the

ARIS Toolset [9]. We provide a formal definition of their semantics in terms of a transition system. This can be further used as a base for behavioral (functional) verification of eEPCs using different model checkers.

Furthermore, we provide a translation to timed colored Petri nets and formulate some correctness criteria for eEPCs that can be checked on the translated eEPCs using CPN Tools.

Future work. For future research, it would be interesting to consider an automatic translation of eEPCs to colored Petri nets and perform verification experiments on large case studies. Since verification by model checking could lead to a blow-up of the state space, reduction techniques would be beneficial (see [8] for an overview). Another line of investigation is the development of a classification of correctness requirements for business processes like it is already done for software processes [14].

References

1. W. van der Aalst. Formalization and verification of event-driven process chains. *Information and Software Technology*, 41(10):639–650, 1999.
2. W. M. P. van der Aalst, J. Desel, and E. Kindler. On the semantics of EPCs: A vicious circle. In *EPK 2002, Proceedings des GI-Workshops und Arbeitskreistreffens (Trier, November 2002)*, pages 71–79. GI-Arbeitskreis Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, 2002.
3. The CPN Tools Homepage. <http://www.daimi.au.dk/CPNtools>.
4. A. Cheng, S. Christensen, and K. Mortensen. Model Checking Coloured Petri Nets Exploiting Strongly Connected Components. In M. Spathopoulos, R. Smedinga, and P. Kozak, editors, *Proceedings of the International Workshop on Discrete Event Systems, WODES96*, pages 169–177, 1996.
5. N. Cuntz and E. Kindler. On the semantics of EPCs: Efficient calculation and simulation. In M. Nüttgens and F. J. Rump, editors, *EPK 2004: Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, Gesellschaft für Informatik, pages 7–26, Bonn, 2004.
6. R. Davis. *Business Process Modeling with ARIS: A Practical Guide*. Springer-Verlag, 2001.
7. J. Dehnert. *A Methodology for Workflow Modeling - From business process modeling towards sound workflow specification*. PhD thesis, TU Berlin, 2003.
8. C. Girault and R. Valk. *Petri Nets for Systems Engineering - A Guide to Modeling, Verification, and Applications*. Springer, 2003.
9. IDS Scheer AG. *ARIS Methods Manual*, 2003.
10. K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Applications*. Springer-Verlag, 1992.
11. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation: Iterative Process Prototyping*. Addison-Wesley, 1998.
12. E. Kindler. On the semantics of EPCs: A framework for resolving a vicious circle. In J. Desel, B. Pernci, and M. Weske, editors, *Business Process Management, BMP 2004*, volume 3080 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2004.

13. P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event-Driven Process Chains. In *19th Int. Conf. on Application and Theory of Petri Nets*, volume 1420 of *LNCS*, pages 286–305. Springer, 1998.
14. G. S. A. Matthew B. Dwyer and J. C. Corbett. Patterns in Property Specifications for Finite-state Verification. In *Proceedings of the 21st International Conference on Software Engineering*, 1999.
15. K. G. Nüttgens and A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). Technical report, Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken, 1992.
16. M. Nüttgens and F.J.Rump. Syntax und Semantik Ereignisgesteuerter Processketten (EPK). In J. Desel and M. Weske, editors, *Promise 2002- Processororientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*, volume LNI, pages 64–77, 2002.
17. A.-W. Scheer. *ARIS : business process modeling*. Springer-Verlag, Berlin, 2nd edition, 1998.

Web Process Dynamic Stepped Extension: Pi-Calculus-Based Model and Inference Experiments

Li Zhang and Zhiwei Yu

School of Software, Tsinghua University, Beijing, 100084, China
lizhang@tsinghua.edu.cn, yzw03@mails.tsinghua.edu.cn

Abstract. Web Processes combine traditional workflow management with Web Services technology. A key challenge to support dynamic composition of Web Processes is to solve the conflicts between process deployment and process execution caused by the inner dependencies. To address this, we have presented a dynamic extension pattern, termed the Web Process Dynamic Stepped Extension (WPDSE). In this approach the process is divided into multiple sub processes, and each sub process is defined and deployed at different times during process execution based on the requirements. A rigorous mathematic modeling language, pi-calculus, is used to define the framework and extension units of the WPDSE. The primary benefit derived from using the pi-calculus is that both the correctness and dynamic performance of the WPDSE model can be effectively verified and analyzed using a mathematically sound approach. This is done using a pi-calculus inference prototype tool called the Interactive Inferring Tool (InferTool).

1 Introduction

Web Processes have a potential for allowing corporations to build inter-organizational business processes. One way to realize Web Processes is by combining of traditional workflow management and Web Services that provide better interoperability in the web-centric world. Such Web Processes represent an inevitable evolution of the pervasive business process management technology [1].

Among the more ambitious architectures for Web Process is that of creating dynamic business processes on the fly. Support for dynamic process requires the capability for dynamic Web services discovery as well as dynamic Web Process deployment. Dynamic discovery of Web services has been proposed in semantic Web service projects like METEOR-S[2]. An approach of semantic Web service composition with the help of semantic templates, which represent abstract capabilities of Web services, has been proposed in [3, 4]. On the fly discovery and integration of services in Web processes leads to creation of dynamic Web processes, which must be modeled for studying dynamism and verifying correctness. This paper deals with mathematical modeling of dynamic processes based using pi-calculus. In order to illustrate a dynamic process, consider the example of an order management process that can dynamically send orders to any number of suppliers based on the order. The number of suppliers is unknown at design time. Hence at design time the number of parallel branches in the process is unknown. Modeling such a process allows us to

study the level of dynamism with respect the number of suppliers. This can help us evaluate system load as well the pattern used for process design.

Previous work on analyzing workflows has used Petri-nets [7,8], but they have been restricted to static workflow patterns. In this paper, we model our system using pi-calculus and show the results of our inference tool. This tool is valuable for not only detecting standard faults like livelock or deadlock, but also for measuring the degree of parallelism a process. In this paper, we will focus on discussing Web Process Dynamic Stepped Extension Model, which contains two parts: Dynamic Stepped Extension Framework Model (DSEFM) and Static Process Model (SPM). This work was done as part of the METEOR-S project at the University of Georgia, which aims to study and model the role of semantics in the complete lifecycle of Web Processes. The BPEL4WS/WSDL based implementation framework will not be discussed in this paper. The interested readers are referred to the literatures [5, 6].

This paper is organized as follows: The motivation scenario is depicted in section 2. In Section 3 we explain the concept of the stepped process and describe the framework of WPDSE in detail. In section 4, the Dynamic Stepped Extension Framework Model based on pi-calculus is discussed. In the section 5, we provide the Static Process Model and show how to construct it. In the section 6, a new graphic representation method is provided to illustrate WPDSE model. In the section 7, the inference tool prototype of the pi-calculus model is described and we will illustrate how to use the inference tool to analyze typical dynamic extension processes. We will review the related work in this area in Section 8. Finally, we present our conclusions and plans for future work in Section 9.

2 Motivation Scenario

Consider a process that supports an electronic product order. The steps involved include the following: (1). receive an electronic product order sent by a retailer; (2). analyze the order and split it in several part items; (3). find out all candidate suppliers for each item; (4). check the compatibility between the different candidate items and select the suppliers; (5).correspondingly deliver a purchase order to each supplier; (6).monitor the status of each order item on a regular basis.

A control flow for such an electronic product order process is shown as Figure 1-b. The number of the parallel branch of the process flows cannot be determined when deploying this process because it is dependent on the result of executing activity “*split order items*”. In addition, finding and binding the Web Services for the activity “*deliver order*” also cannot be completed because the suppliers for the item are unknown before checking the compatibility of the part items.

The approach we use to address this conflict is to divide the whole process into three smaller processes: one top process and two sub processes, and follow this up by deploying the top process. As shown in Fig. 1-a, the top process consists of four activities. When deploying, only two activities, “*receive order*” and “*check compatibility*” can be bound to their corresponding Web Services, the others cannot be deployed. After executing activity “*receive order*”, the following activity, “*find suppliers*”, can be extended. At this time, all the activities in this sub process can be

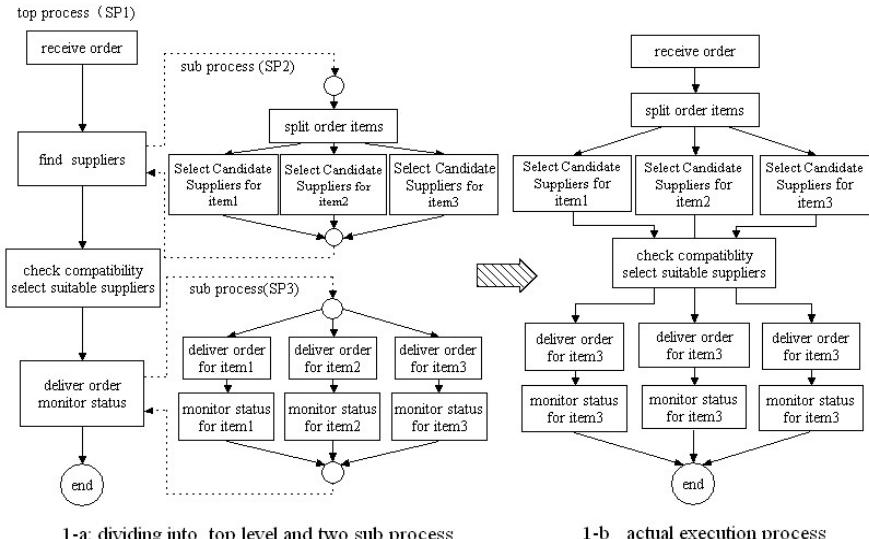


Fig. 1. The electronic product order process

deployed. Similarly, after the activity “*check compatibility*” is completed, the activity “*delivery order and monitor status*” can be extended and all the activities in this sub process can be deployed.

3 The Framework of WPDSE

In this section, we will describe the overall framework of WPDSE. A high level view of WPDSE framework is shown as Fig. 2. It is composed of four components as following:

Process Template Manager (PTM): An abstract process template contains some semantic and structure information. We call abstract process template as static process model, which is used as a basic extension unit. The functions of *Process Template Manager* are to produce and manage the process templates. On the initial status, *Process Template Manager* receives the request from the START and creates the request for the first *Process Template* (PT), which is the top process of all the sub processes. After that, *Process Template Manager* will constantly receive the process template request through the feedback channel from the *Process Interaction Engine* (PIE) and create a process template sequence: $\{PT_1, PT_2, \dots, PT_n\}$.

Process Deployment System (PDS): The *Process Deployment System* receives the process templates PT_i from the *Process Template Manager* and translates them into executable processes according to the semantic and structure information contained in process templates.

Process Execution Engine (PEE): *Process Execution Engine* receives executable processes from *Process Deployment System* and invokes them as a new static process SP_i . In the *Process Execution Engine*, there is a static process pool, which is a set of

static processes SP_i and its initial status is null. The first element in the static process pool is the top process. The number of the static processes in the static process pool is constantly changing. New extended static processes are added to the static process pool and the finished static processes are removed from the static process pool. Only after the static process pool becomes null again, the whole process is terminated.

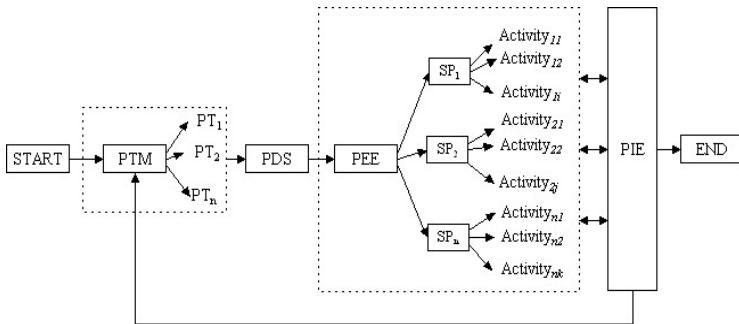


Fig. 2. The framework of WPDSE

Activity is one of basic elements of the static process. In the framework of WPDSE, each activity contains an extension property “*activityType*” which is used to judge if current activity needs to be extended. If an activity needs not to be extended, it will be directly bound with a Web Service. Otherwise, when this activity is executed, new sub process should be extended.

Process Interaction Engine (PIE): Although the activities contain the information for judging if current activity needs to be extended, the extension operations are not implemented at the activity level. All the interaction operations defined in the activity must be processed by the *Process Interaction Engine*. If an activity needs to be extended, *Process Interaction Engine* will send an extension request to *Process Template Manager* by the feedback channel.

4 Dynamic Stepped Extension Framework Model

In the late 1980s, Robin Milner developed a formal theory of mobile processes: pi-calculus [9], which has two aspects. First, it is a theory of mobile systems, which provides a conceptual framework for understanding mobility. Second, it is a mathematical model of process whose interconnections change as they interact. In this paper, we use standard pi-calculus syntax to model WPDSE.

Dynamic Stepped Extension Framework Model (DSEFM) is a pi-calculus based model which is only used to describe interaction behaviors between the components of the framework of WPDSE given in the Fig. 1. DSEFM consists of four independent and parallel agents or engines which are modeled by pi-calculus. So, according to the theory of pi-calculus, as soon as the DSEFM is launched, all the agents or engines in it are simultaneously invoked and run in parallel. At the initial

status, each agent or engine in the DSEFM contains only one transactional process to monitor its special channels for receiving the input requests. Once an input request arrives, this transactional process immediately deals with it, and at same time, a new transactional process is forked for the next request.

$$\text{DSEFM} ::= \text{CA} \mid \text{PTIE} \mid \text{PEE} \mid \text{PIE} \quad (1)$$

4.1 ClientAgent (CA)

CA defines the basic interaction behaviors implemented in START and END components in the framework of WPDSE. CA is an initiator of the whole Web Process. In CA model, the requestPT_1 is an input request, created by a client, for the first Process Template. The inputPT_1 is an input related to launch a new Web Process. The returnPT_1 is a mobile channel which is created by CA and will be transmitted to the executed static process (SP_1). The returnPT_1 is a key parameter for implementing dynamic extension because it helps to dynamically establish the connection between CA and SP_1 . All the messages above are sent to PTIE through a common channel $chlPT$. Finally, CA monitors the channel “ returnPT_1 ” for receiving the results of the whole process execution.

$$\begin{aligned} \text{CA} ::= & (v \text{ requestPT1}, \text{returnPT1}, \text{inputPT1}) \\ & \text{chlPT} < \text{requestPT1}, \text{returnPT1}, \text{inputPT1} >. \text{returnPT1} (\text{result}) \end{aligned} \quad (2)$$

4.2 Process Template Instantiation Engine (PTIE)

We integrate the dynamic behaviors in both PTM and PDS components in the framework of WPDSE into the PTIE model. In order to emphasize the interaction features in WPDSE, we ignore those complex logics related to process template design and Web Services discovery. These complex logics are abstractly represented as two inner transaction processes: PTM and PDS.

The identifier “i” indicates that $\text{PTIE}(i)$ is the i^{th} transaction process which is responsible of dealing with the i^{th} process template PT_i . The $chlSP_i$ is a new channel which is dynamically created for transmitting process template PT_i .

$\text{PTIE}(i)$ first receives the process template request requestPT_i on the channel $chlPT$, which is a common channel and can be used by multiple different transaction process. For example, If $i=1$, the channel $chlPT$ is used as input channel between CA and PTIE. Otherwise, the $chlPT$ is used as a feedback channel between $\text{PIE}(i)$ and $\text{PTIE}(i)$. After $\text{PTE}(i)$ receives the PT_i ’s request, two inner transaction processes, PTM and PDS, will be invoked in order. Their output is an executable static process SP_i . Finally, $\text{PTE}(i)$ will forks three parallel transaction process: (1).sending the SP_i to PEE by the channel add ; (2).transmitting inputTP_i to SP_i through channel $chlSP_i$; (3).launching a new PTIE’s transaction process $\text{PTIE}(i+1)$.

$$\begin{aligned} \text{PTIE}(i) ::= & (v \text{ chlSP}_i \text{ chlPT}(\text{PT } i, \text{returnTP}_i, \text{inputTP } i) . \\ & (\text{PTM. PDS} . (\text{add} < SP_i > \mid \text{chlSP}_i < \text{inputTP } i > \mid \text{PTIE}(i+1))) \end{aligned} \quad (3)$$

4.3 Process Execution Engine (PEE)

In pi-calculus, a process also can be transmitted as a parameter from one location to another and be invoked on its target location. In DSEFM, the variable SP is one executable process which is send as a parameter from $PTIE$ to PEE and invoked by PEE in static process pool.

$$PEE(i) ::= add(SP).(SP \mid PIE(i+1)) \quad (4)$$

$PEE(i)$ is the i^{th} transaction process of PEE . It receives i^{th} static process SP_i and invokes it. At the same time, it launches a new $PEE(i+1)$ transaction process to deal with the next static process SP_{i+1} . All the static processes are run in parallel. Like traditional workflow process, each SP_i consists of several basic elements. The details about the Static Process Model (SPM) and its basic elements are discussed in next section. Here, we only take a simple example to explain relationships between DSEFM and SPM.

Suppose that SP_1 is a sequence static process which only consists of three basic elements: one “Entry” element, one “Activity” elements and one “Exit” element. The SP_1 ’s Model is shown as following:

$$\begin{aligned} SP_1 (chlSP1, returnTP1) ::= & \text{Entry(chlSP1,.chlActA)} \\ & \mid \text{Activity(chlActA, sendResult)} \\ & \mid \text{Exit} \square \text{sendResult} \square \text{returnTP1} \square \end{aligned} \quad (5)$$

SP_1 has two input parameters: one is channel $chlSP_1$ which connected with $PTIE$; another is also a mobile channel $returnTP_1$, which originally comes from CA or PAE . If the SP_1 is a top process, the channel $returnPT_1$ is used to connect with CA and to transmit the final results of the whole process. Otherwise, it is linked to PIE for returning the result of the sub process.

The models of three types of basic elements contained in SP_1 are provided as following to illustrate how to extend a new sub process during executing SP_1 .

$$\text{Entry} ::= \text{chlSP1 (input). chlAct<input>} \quad (6)$$

$$\begin{aligned} \text{Activity(chlAct,sendResult)} ::= & (v \text{ activityInfo, returnAct, type}) \\ & \text{chlAct (input).chlPAE<activityInfo, type, inputAct, returnAct>} \\ & \text{returnAct(resultAct).sendResult<resultAct>} \end{aligned} \quad (7)$$

$$\text{Exit(sendResult, returnPT)} ::= \text{sendResult(result).returnTP <resultSP>} \quad (8)$$

Like traditional workflow process, we limits each SPM has only one entry and exit point. In this example, “Entry” is the start point. It receives input parameters through channel $chlSP_1$ and then transfers them to its next node through channel $chlAct$.

Every *Activity* in the SPM has four channels: $chlAct$, $sendResult$, $returnAct$ and $chlPAE$. First three of them are mobile channels. The channel $chlAct$ and $sendResult$ are inputted as parameters and respectively used to connect with previous and next node of the current Activity. The channel $chlPAE$ is a common channel which is specially used to communicate with PIE for transmitting the messages related to sub process extension. The channel $returnAct$ is created in current *Activity* and also used to connect PIE for receiving the results after the current activity is finished.

4.4 Process Interaction Engine (PIE)

All the *Activities* are managed by *PIE* and each *Activity* is associated to a *PIE* transaction process. For example, *PIE(i)* is the i^{th} transaction process to deal with the i^{th} *Activity*. The connection between *Activity* and *PIE* are established through channels *chlPAE* and *returnAct*. The *chlPAE* is common channel used to receive the input requests. The *returnAct* is a mobile channel which is transmitted through the channel *chlPAE*. In SPM, there are two types of activities: Extensible Activity (EA) and Non-Extensible Activity (NEA). If the type of an *Activity* is non-extensible, an inner transaction process “*Exec*”, which is probably associated a Web Service, is activated. Otherwise, this *Activity* should be extended. In this case, a request related to sub process template *requestPT_i* and a new mobile channel *returnPT_i* is created and another inner transaction process “*Exec1*” is executed. Then, both *requestPT_i* and *returnPT_i* are sent to *PTIE* through the common channel *chlPT*. Finally, no matter what type the current activity is, the results of the activity execution are returned back through the channel *returnAct*.

$$\begin{aligned} \text{PIE (i)} ::= & (v \text{ resultAct}) \text{ chlPAE(activityInfo, type, inputAct, returnAct)}. \quad (9) \\ & ([\text{activityType} = \text{EA}] (v \text{ requestPT}_i, \text{returnPT}_i, \text{inputPT}_i) \text{ Exec1}. \\ & \quad \text{chlPT} < \text{PT}_i, \text{returnPT}_i, \text{inputPT}_i >. \\ & \quad \text{returnPT}_i (\text{resultPT}_i). \text{Exec2.returnAct} < \text{resultAct} > \\ & + [\text{activityType} = \text{NEA}] \text{ Exec.returnAct} < \text{resultAct} >) \\ & | \text{ PIE}(i+1) \end{aligned}$$

5 Static Process Model

There are two necessary conditions to dynamically compose Web Processes. The first is the supporting mechanism to guarantee that various transaction processes in Web Process lifecycle can communicate effectively and work collaboratively. The second is the extension units used to extend the Web Process step by step. In this section, we will discuss the extension units of the WPDSE: Static Process Model (SPM).

5.1 The Basic Elements

We have defined 12 basic elements which are used to construct the diversity of the SPMs, including *Activity*, *Entry*, *Exit*, *Connector*, *ParallelSplit*, *Synchronization*, *ExclusiveChoice*, *SimpleMerge*, *MultiChoice*, *SynchronizingMerge*, *MultiMerge*, *NoutofMjoin* etc. All the basic elements have been modeled based on pi-calculus. Each basic element will be run as an independent process. The relationships between them are established by their common channels. Because of the limitation in length, we can not explain all the basic element’s models. The models of the first three basic elements have been introduced in the previous section. In this section, we will illustrate the additional two basic elements: *ParallelSplit* and *Synchronization*. The detailed descriptions of the remainder element can be found in technical report [6].

ParallelSplit has one input channel “*receive*” and several output channels “*send₁*, ..., *send_n*”. *ParallelSplit* first gets the “*input*” on the channel “*receive*” and then outputs the “*input*” on all the channels “*send₁*, ..., *send_n*” simultaneously.

ParallelSplit (*receive*, {*send*₁, ..., *send*_n}) ::= (10)
receive (*input*).(*send*₁< *input*₁>| *send*₂< *input*₂>| ... | *send*_n<*input*_n>)

Synchronization has several input channels “*receive*₁, ..., *receive*_n” and only one output channel “*send*”. *Synchronization* can get different messages “*input*₁, ..., *input*_n” from different input channels “*receive*₁, ..., *receive*_n” simultaneously. After all input messages are received and integrated by inner process Integration, the “*result*” is created and outputted to the channel “*send*”.

Synchronization ({*receive*₁, ..., *receive*_n}, *send*) ::= (11)
(v *result*)(*receive*₁(*input*₁) | *receive*₂(*input*₂) | ... | *receive*_n(*input*_n)).
Integration. *send*<*result*>

5.2 The Example of SPM

We still take the electronic product order process described in section 2 as an example to explain how to construct the SPM. According to Fig. 1-a, the electronic product order process should consist of three SPMs: *SP1*, *SP2* and *SP3*. The pi-calculus models of three SPMs and the extension process diagram are shown in the Fig. 3.

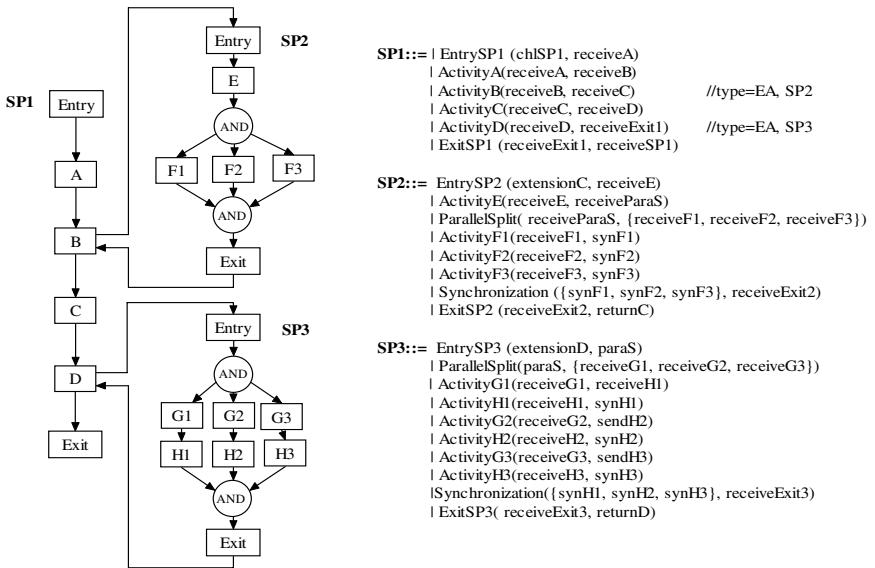


Fig. 3. The SPM of the electronic product order process

SP1 is top process which is sequence process and includes four activities: *receive order* (*A*), *find suppliers* (*B*), *Check Compatible* (*C*), *deliver order & monitor status* (*D*). We suppose that the activities *B* and *D* are extensible activity.

SP2 is the sub process which is extended by activity *B*. It contains four activities: *Split Order Items* (*E*) and three parallel activities (*F1*, *F2* and *F3*).

SP3 is the sub process which is extended by activity D. It also contains three parallel sequence processes which respectively include six activities: *Place Order&Monitor Status for Item1 (G1 and H1), for Item1 (G2 and H2), and for Item1 (G3 and H3)*.

6 A Graphic Representation of the WSDSE Model

Pi-calculus is a block-structured formal modeling language. Although it is very convenient to express the operation semantics for concurrence processes with evolving capabilities and is easy to be interpreted by machines, this kind of block-structured modeling language is very difficult to be understood by human being, especially for the complex models like the WSDSE Model. Unfortunately, there are few investigations on this problem. As much as we know, Joachim Parrow designed an Interaction Diagram to graphically represent concurrent process. In [22], he explained how to mapping the primitive construct of pi calculus into the graphic counterparts and how to build pi calculus model from top to down in the graphic way. Although Joachim Parrow's Interaction Diagram can efficiently describe interaction behavior among multiple mobile agents, it is not convenient to express the evolutions of multiple lifecycle processes.

We have designed a new graphic representation method which can visualize both the mobility and evolution of the WSDSE Model. We call it as Swimming Lane Interaction Diagram (SLID). The conception of SLID comes from UML Activity Diagram. In SLID, the evolution of process lifecycle can be clearly spread along with swimming lanes. The interaction behaviors among multiple processes can be shown by the directed connecting lines between the swimming lanes. We draw a SLID by mapping the algebraic symbols of pi calculus model into graphic symbols in SLID. Table 1 provides these mapping relations and rules.

Table 1. The mapping relations between the algebraic and graphic symbols

No.	Pi calculus	SLID	No.	Pi calculus	SLID
1	Agent or Engine	Swimming Lane	6	•	
2	Transaction Process	Activity Diagram	7	+	○+○
3	Inner Process	[P]	8		○ ○
4	$x < a >$	$\diamond a \rightarrow$	9	(vx)	↓(x)
5	$x (a)$	$a \leftarrow \circlearrowleft$	10	0	▨

We still take the electronic product order process as an example how to use the SLID to illustrate the mobile and extension characteristics of the WSDSE Model. Because the SLID of the electronic product order process is too complex to express the whole extension process in one picture, we only give the part of them in Fig. 4.

The whole SLID of the electronic product order process contains seven swimming lanes. The first lane is for the process CA, which has only one transaction process.

The second lane is for the process *PTIE*, which should creates three transaction processes, *PTIE(1)*, *PTIE(2)* and *PTIE(3)*. They are respectively used to deal with *SP₁*, *SP₂* and *SP₁*. Next lane is for the process *PEE*, which also invokes three transaction processes, *PEE(1)*, *PEE(2)* and *PEE(3)*. The fourth lane is for all the static processes, including *SP₁*, *SP₁* and *SP₂*. The fifth lane is for “Elements” which are contained in the static processes. For example, the top process *SP₁* consists of six elements, including *EntrySP1*, *ActivityA*, *ActivityB*, *ActivityC* *ActivityD* and *ExitSP1*. The last lane is for process *PIE*. Because each *Activity* must be independently associated with a process *PIE*, there are totally 14 process *PIEs* in the Fig. 4, but we only draw two of them: *PIE(1)* and *PIE(2)*, which respectively deal with *ActivityA* and *ActivityB*.

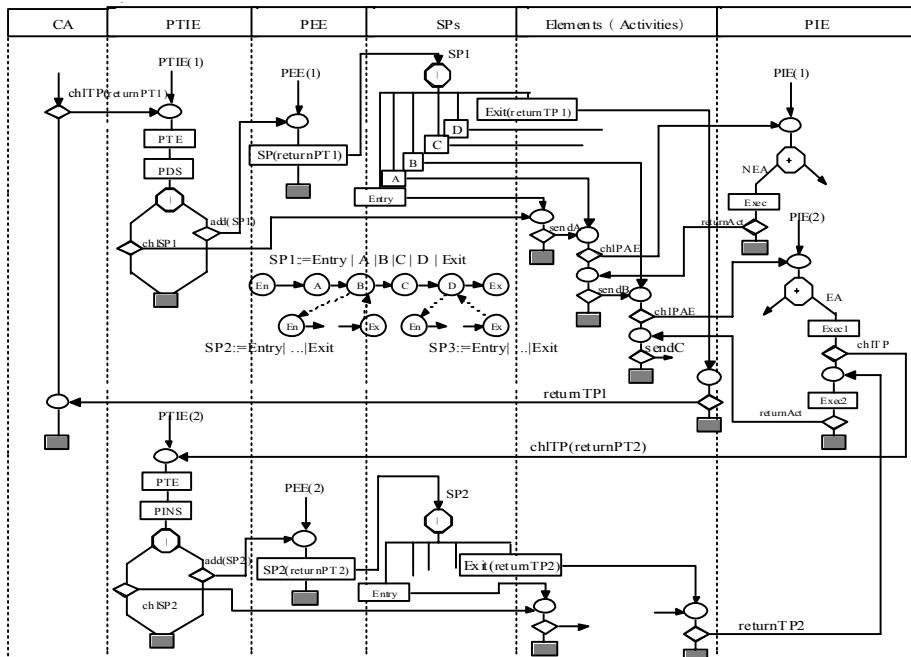


Fig. 4. The SLID of the electronic product order process model

From the Fig. 4, a lot of the mobile and extension characteristics of the WSDSE Model are exhibited clearly. For example:

- The process *CA* is both the start and end points of the whole electronic product order process.
- The channel *returnPT1* is transmitted in the whole process, firstly from *CA* to *PTIE(1)*, again to *PEE(1)*, then to *SP1*, next to *Exit*.
- The top process *SP1* is firstly created in the *PTIE(1)* and then transmitted as a parameter from *PTIE(1)* to *PEE(1)* and invoked in there.

- Activity B in the *SP1* communicates with the process *PIE(2)* through the channel *chlPAE*. Because it is an extensible activity, the *PIE(2)* must send a process template request *returnTP2* to the process *PTIE(2)* by the channel *chlTP*. Then, *PTIE(2)* creates sub process template *SP2*, which is invoked in process *PEE(2)*.

7 Inference Experiments Based on Pi Calculus

There have been a few tools [10] which have been used to infer the pi-calculus model, but these tools are unsuitable to infer the WPDSE model because they are generally designed for deterministic pi-calculus models. The WPDSE model is undetermined before starting inference because the SPMs in the WPDSE model must be selected during inferring. In order to simulate and analyze the WPDSE model, we have developed a prototype of Interaction Inferring Tool (InferTool) which can provide the interaction portal for supporting to dynamically select the different sorts of SPMs.

7.1 The Architecture and Principle of the InferTool

The InferTool architecture is shown in Fig. 5. It contains five essential components: *InferringEngine*, *ModelEditor*, *InteractionPortal*, *Analyzer* and *ModelBase*.

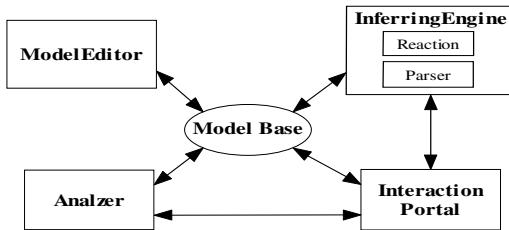


Fig. 5. The architecture of the InferTool

The heart of the InferTool is *InferringEngine* which is used to simulate the process evolution by pi-calculus inference. The basic principle of the InferTool is to automatically perform a series of *Reaction Operations* according to the rigorous *Prefix Reductions* rules which include two steps: finding a pair of matched *Prefix* and executing *Reduction*. For example, suppose a process $P ::= \{ x(y) | x(u).u(v) \}$. Because $x(y)$ and $x(u)$ are a pair of matched prefixes, the channel x is reducible. The result of the reduction operation is that the channel x is removed from process P and name u is replaced with name y . Finally, process P will evolve into a new process $P' ::= \{ y(v) \}$. The *Reaction Operation* can be represented as following:

$$P ::= \{ x(y) | x(u).u(v) \} \xrightarrow{\tau} P' ::= \{ y(v) \} \quad (12)$$

The InferTool provides an interaction environment to select the different sorts of SPMs. Before performing the inference experiments, we first need to input the DSEFM and SPM model files through *ModelEditor* and store them into the

ModelBase. Then, we must design typical test cases which consist of the different sort of SPMs. We have performed the following three kinds of inference experiments: simulation experiments and faults-discovering experiments.

7.2 Simulation Experiments

The simulation experiments can help us understand WPDSE framework and verify the correctness of the WPDSE model by observing the whole extension processes of the WPDSE and recording the performance values created during executing. We have performed a great deal of experiments which proved that the WPDSE model can correctly and effectively represent the extension characteristics of dynamic Web process. On the following, we still take the electronic product order process as test case to illustrate inferring process based on the InferTool. The pi-calculus model of the electronic product order process has given in the Fig. 3.

The Table 2 shows the part of the records of inferring process. In the table 2, the column “No” represents the sequence number of the reduction. The column “reduction location” represents the location of the channel which links two current parallel processes. The column “reduction type” represents the type of the reductions. There are three type reductions, including *channel*, *condition* and *inner transaction process*. If the type of reduction is *channel*, an alpha conversion will be executed. In that case, the columns “Sent Name” and “Received Name” respectively record the parameters to be sent or received by the reductions. The last two columns in the table record the status of the system related to the reduction. The column “CCPN” means the number of the current parallel processes which shows the instantaneous load of the inferring system. Column “EPN” represents total number of executed processes.

For example, the reduction in the first row of the table 2 is involved into two processes: *CA* and *PTIE(1)*. The parameters *PT1*, *returnP1* and *inputP1* are sent by channel the *chlPT* from process *CA*. At same time, process *PTIE* receives respectively those t parameters as input parameters: *PT*, *returnP* and *inputP*. After first reduction has finished, CCPN is 5 and EPN is 0.

For the electronic product order process, there are total 149 reductions. Table 2 only lists three key parts of the reductions, including initial reduction, activity B extension reductions, end reductions.

The reductions from the row 1 to the row 6 are the initial reductions which include four channel reductions and two operation reductions. Firstly, the process *CA* sends initial parameters to the process *PTIE* by channel the *chlPT*; Secondly, the process *PTIE* sends the top process *SPI* to the process *PEE* by the channel the *add*; Then, the process *PEE* invokes the top process *SPI*. After that, process *PTIE* sends the parameter *inputP1* to the process *EntrySPI* by channel *chlSP*. Finally, the activity *ActivityA* is invoked by receiving a message *inputP1* through the channel *recieveA*.

The reductions between row 14 and row 21 describe that the Activity B extension reductions. The activity *ActivityB* is invoked by the activity *ActivityA* by channel the *sendB*. The activity *ActivityB* sends the massage *activityInfoB*, *APT*, *output* and *returnActB* to the process *PIE* through channel *chPAE*. Because the type of the Activity *ActivityB* is *EA*, the *PIE* invokes an inner operation *Exec1* and communicates with the process *PTIE* through the channel *chlPT*. Finally, the new static process *SP2* is send to the *PIE* from the process *PTIE* through channel *add* and is invoked in *PIE*.

Table 2. The part of the reduction records of an inferring experiment

N o.	Reduction Location	Reduction Type	Reduction Name	Sent Name	Received Name	CCPN	EPN
initial reduction,							
1	CA, PTIE	Channel	chlPT	PT1, returnP1, inputP1	PT, returnP, inputP	5	0
2	PTIE	Operation	PTM, PDS	-	-	5	0
3	PTIE, PEE	Channel	add	SP1	SP	7	0
4	PEE	Operation	SP1	-	-	7	1
5	PTIE, EntrySP1	Channel	chlSP1	inputP1	input	7	1
6	EntrySP1, ActivityA	Channel	sendA	inputP1	inputActA	14	2
activity B extension reductions							
14	ActivityA ActivityB	Channel	sendB	resultActA	inputActA	14	5
15	ActivityB	Operation	*Analysis	-	-	14	6
16	ActivityB, PIE	Channel	chlPAE	activityInfoB, APT, output, returnActB	activityInfo ,	14	6
17	PIE	Condition	[type = EA]	-	-	15	6
18	PIE	Operation	*Exec1	-	-	15	6
19	PIE, PTIE	Channel	chlPT	PT2,returnP2, inputP2	PT,returnP, inputP	15	6
20	PTIE, PEE	Channel	add	SP2	SP	17	6
21	PEE	Channel	chlSP2	inputP2	input	17	7
end reductions							
14 7	PIE, ActivityD	Channel	returnActD	resultActD	resultActD	30	57
14 8	ActivityD ExitSP1,	Channel	sendExit1	resultActD	result	29	58
14 9	ExitSP1, CA	Channel	returnTP1	result	result	28	59

The reductions between row 147 and row 149 show the ending process of the electronic product order process. After the reduction 147 is executed through the channel *returnAct* between *PIE* and *ActivityD*, the sub process *SP3* ends and return to the activity *ActivityD*. The result of the reduction 148 is that the top process is evolved from *ActivityD* to *ExitSP1*. Finally, after the reduction 149 is executed, the whole top process *SP1* is finished and the result is return to the process *CA*.

7.3 Fault-Exposing Experiments

The purpose of the fault-exposing experiments is to prove that the unreasonable structure fault in the SPMs can be discovered through pi-calculus inference. Two typical structure faults, the deadlock and livelock, will be discussed below. Through these experiments, the basic method to discovery faults can be explained.

Deadlock

In WPDSE model, deadlock is caused due to the lack of the matched prefix. For a reasonable structure SPM, all the prefixes are finally matched with their complementary prefixes. So, when an inferring process ends, all the prefixes are reduced and all the processes are executed. Contrarily, if there are the deadlocks in the SPM, some prefixes will not be reduced. As a result, the inferring process will be terminated and some unreduced prefixes and unexecuted processes will be left. So, the methods to judge the deadlock is to see if there are the unmatched prefixes or unexecuted processes left after the inferring process is stopped. Let us take a simple example, shown in Fig. 6, to explain the deadlock.

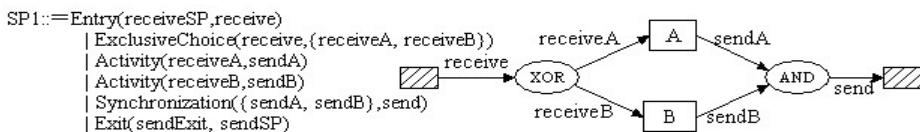


Fig. 6. The test case for deadlock

When the experiment is stopped, the following prefixes and processes are left:

Activity(receiveB,sendB) | Exit(send,sendSP) | {sendB(input2).send<output>} (13)

We can find that the reason of terminating the inferring process is that the InferTool can not discovery the pair of the complementary prefixes of the channel *sendB*. So we conclude that there certainly is a deadlock in the test case.

Livelock

The reason of creating livelock is that there exist cycle loop in the process control flow. So, if there are livelock in SPM, some reduction operation will be executed repeatedly and the inferring process will never be stopped. We also take a simple example, shown in the Fig. 7, to illustrate how to judge the livelock.

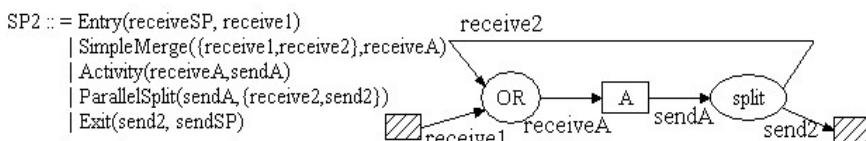


Fig. 7. The test case for livelock

In this experiment, we found that three reduction operations, *receive2*, *receiveA* and *sendA*, are executed repeatedly, as shown in Table 3. Apparently, there is a livelock in this SPM and the cycle loop of the livelock contains the channel *receive2*, *receiveA* and *sendA*.

Table 3. The part of the records of the test case for livelock

No.	Reduction Name	Sent Name	Received Name
10	sendA	resultAc	Input
11	receive2	output1	Input
12	receiveA	Output	inputAc
14	sendA	resultAc	Input
15	receive2	output1	Input
16	receiveA	Output	inputAc
18	sendA	resultAc	Input

8 Related Works

The main focus of our paper is to study the dynamic composition method of Web Process and pi calculus based modeling and analysis. To the best of our knowledge, there is no research effect of this problem before. However, there have been many research literatures and commercial tools which are closely related to our works. We can conclude these researches as following three aspects:

8.1 The Composition of Dynamic Business Processes

There have been a lot of researches on systems for supporting to dynamic workflow processes. For example, the literature [11] proposes a dynamic workflow system that is capable of supporting non-deterministic processes such as those found in collaborative product design scenarios. In the paper [12], Fakas GJ presents the architecture of a novel Peer to Peer (P2P) workflow management system, which employs P2P principle to simplify the workflow process and provide a more open, scalable process model that is shared by all workflow participants.

However, the dynamic process discussed in our paper is Web Process. The most of researches on the dynamic composition of Web Process is mainly emphasize particularly on the utility of Web Service. The representative research works are introduced in the literature [13, 4]. The paper [13] investigates how to efficiently discover Web services-based on functional and operational requirements, and how to facilitate the interoperability of heterogeneous Web services. The paper [4] focuses on automatically discovering web services and binding them with the activities in process template. Our research works are built on the basis of these research achievements. We are mainly concentrated on another important aspect of the Web Process: the dynamic composition of the whole processes.

8.2 The Modeling and Analyzing of Dynamic Business Processes

The kernel problem discussed in our paper is modeling and analyzing of dynamic business processes. A lot of research has been done on formal modeling for workflow process definitions. Petri nets is a widely interested and applied formal modeling language [7,8]. Apart from Petri nets, other researchers have proposed alternatives, for example, Eshuis[14] proposes to use the Unified Modeling Language; Only a few

papers in the literature focus on the verification of workflow process definitions. In the paper [15], Yamaguchi discussed the formal modeling and performance evaluation for the dynamic change of workflow and proposed a measure, to evaluate dynamic change of workflows.

A remarkable investigation is reported in paper [16] which proposes the inheritance-preserving transformation rules to avoid some problems which occur during transforming a process instance from an old process definition to a new one. Four inheritance relations are defined and a tool to support for the inheritance notions is developed. The distinct differences between our research and those researches above are that our dynamic changes is the lifecycle extension of the whole process from one lifecycle to another, not the changes of process definitions or structures from one process definition to another.

8.3 The Pi-Calculus Based Modeling and Inferring

Today the pi calculus and related theories make up a large and divers field with hundreds of published papers. Most papers about the pi calculus focus on variants of the language and proofs of their correctness. In recent years, the theory and method related to pi calculus have been obviously focused on the various area of applications [17, 18]. We also found some research reports related to Web services. Bhargavan K, in the paper [19], introduces a language-based model for WS-Security protocols and this model is embedded with the applied pi calculus for verifying authentication properties. The literature [20] introduces a peer-to-peer model based on pi calculus for reasoning about the dynamic behaviors of web data.

There are only a few academic reports on pi calculus based model of business process, especially for the Web Process. In the paper [1], Howard Smith expatiates his opinions about the pi calculus's application in the business process management. He considers that pi calculus can provide a way of modeling for a wide variety of process management systems. The paper [21] is one of early literatures which studied to model dynamic business process really based on pi calculus. This paper provides an educational exercise in using the pi calculus to model a dynamic business process which includes multiple agents of an electronic fish marketplace. Although the pi calculus based model provided in this paper is aimed at a special business process in fish marketplace, the results of the investigation has made it clear that the pi calculus might provide a suitable basis for defining the interaction behavior of dynamic business processes.

9 Conclusions and Future Works

We propose a dynamic extension pattern: WPDSE. The basic conception of the WPDSE is that a whole process is divided into multiple sub processes, and each sub process is defined and deployed in different time according the requirements created during process execution. From the perspective of the control theory, WPDSE is a close loop control system where there is a feedback channel from the interaction stage to the design stage. We select a pi calculus, as modeling language to formally express the whole WPDSE model. The basic dynamic characteristic of the WPDSE is

mobility which is exhibited in two aspects: process mobility and channel mobility. Our investigation has proven that pi calculus is an ideal model language to describe this kind of mobile features in the WPDS model. As shown by our results, pi-calculus based modeling can be used not just for fault detection like deadlocks and livelocks, but also for studying the dynamism of Web processes. Future research directions that we are beginning to explore include the following:

- At present, there have been a few industrial specifications which are widely used to model Web Process, such as WSCDL, WSCI, BPML and BPEL4WS. It is certainly valuable to investigate the relationship between pi calculus and current main industrial specifications to analyze the performance of the process.
- We will continue to study other dynamic composition patterns or methods to meet different application requirements, such as more complex dependency relations.
- The evaluation method for dynamic business process should be different from traditional static business process, especially for Web Process. We will study the metric methods for estimating the dynamic performance of the Web Process.
- Continue to research and improve InferTool so that it can support the full pi calculus syntax and provide the stronger functions for analyzing dynamic performances of pi calculus models.

Acknowledgement

The work described in this paper was partially supported by The National Basic Research Program of China (Grant No. 2002CB312006) and the National High-Tech R&D Program of China (Grant No.2003AA411022).

References

1. Howard Smith, Business process management—the third wave: business process modeling language (BPEL) and its pi calculus foundations, *Information and Software Technology* 45(2003), 1065–1069.
2. Abhijit Patil, Swapna Oundhakar, Amit Sheth, Kunal Verma, METEOR-S Web service Annotation Framework, Proceeding of the World Wide Web Conference, July 2004.
3. Sivashanmugam, K., Verma, K., Sheth, A., Miller, J., Adding Semantics to Web Services Standards , Proceedings of the 1st International Conference on Web Services (ICWS'03), Las Vegas, Nevada (June 2003) pp. 395 – 401.
4. Sivashanmugam, K., Miller, J., Sheth, A., Verma, K., Framework for Semantic Web Process Composition, Special Issue of the International Journal of Electronic Commerce (IIEC), Eds: Christoph Bussler, Dieter Fensel, Norman Sadeh, Feb 2004.
5. Verma, K., Akkiraju R., Goodwin R., Doshi P., Lee J., On Accommodating Inter Service Dependencies in Web Process Flow Composition, AAAI Spring Symposium 2004.
6. Li Zhang, Zhiwei Yu, Kunal Verma, Amit P. Sheth, Dynamic Business Process Modeling based on Pi Calculus, Technical Report, LSDIS Lab. Computer Science Department, The University of Georgia, 2004.
7. W.M.P. van der Aalst. Three good reasons for using a Petri-net-based workflow management system. In S. Navathe and T. Wakayama, editors, International Working Conference on Information and Process Integration in Enterprises (IPIC'96), pages 179–201, Cambridge, Massachusetts, USA, November 1996.

8. W.M.P. van der Aalst and K.M. van Hee. Workflow Management: Models, Methods, and Systems, MIT Press, Cambridge, MA, 2002.
9. R. Milner. Communicating and Mobile Systems: the Pi calculus. Cambridge University Press, 1999.
10. Jérôme Feret, p.s.a.: A p-Calculus Static Analyzer, <http://move.to/mobility>.
11. Zeng LZ, Flaxer D, et al. PLMflow-dynamic business process composition and execution by rule inference, LECTURE NOTES IN COMPUTER SCIENCE 2444: 141-150 2002.
12. Fakas GJ, Karakostas B, A peer to peer (P2P) architecture for dynamic workflow management, information and software technology 46 (6): 423-431 MAY 1 2004.
13. Cardoso, J. and A. Sheth, "Semantic e-Workflow Composition," Journal of Intelligent Information Systems (JIIS), 21(3): 191-225, 2003.
14. H. Eshuis, Semantics and Verification of UML Activity Diagrams for Workflow Modelling, PhD thesis, University of Twente, Enschede, The Netherlands, October 2002.
15. M. Schroeder. Verification of business processes for a correspondence handling center using CCS. In EUROVAV, pages 253–264, 1999.
16. W.M.P. van der Aalst, Inheritance of Dynamic Behaviour in UML. In D. Moldt, editor, Proceedings of the Second Workshop on Modelling of Objects, Components and Agents (MOCA 2002), volume 561 of DAIMI, pages 105-120, Aarhus, Denmark, August 2002.
17. B. C. Pierce and D.N. Turner. Pict: A programming language based on the pi calculus, Technical Report 476, Indiana University, March 1997.
18. Curti M, Degano P, Priami C, et al. Modelling biochemical pathways through enhanced pi calculus, THEORETICAL COMPUTER SCIENCE 325 (1): 111-140 SEP 28 2004.
19. Feret J, Confidentiality analysis of mobile systems , LECTURE NOTES IN COMPUTER SCIENCE 1824: 135-154 2000.
20. Gardner P, Maffeis S, Modelling dynamic web data , LECTURE NOTES IN COMPUTER SCIENCE 2921: 130-146 2004.
21. Julian Padget and Russell Bradford, A Pi Calculus Model of a Spanish Fish Market-- Preliminary Report□<http://www.iiia.csic.es/amet98/abstract14.html>.
22. Joachim Parrow, Interaction Diagrams, Nordic Journal of computing 2(1995), 407-443.

Petri Net + Nested Relational Calculus = Dataflow

Jan Hidders¹, Natalia Kwasnikowska², Jacek Sroka³, Jerzy Tyszkiewicz³,
and Jan Van den Bussche²

¹ University of Antwerp, Belgium

² Hasselt University, Belgium

³ Warsaw University, Poland

Abstract. In this paper we propose a formal, graphical workflow language for dataflows, i.e., workflows where large amounts of complex data are manipulated and the structure of the manipulated data is reflected in the structure of the workflow. It is a common extension of

- *Petri nets*, which are responsible for the organization of the processing tasks, and
- *Nested relational calculus*, which is a database query language over complex objects, and is responsible for handling collections of data items (in particular, for iteration) and for the typing system.

We demonstrate that dataflows constructed in hierarchical manner, according to a set of refinement rules we propose, are *sound*: initiated with a single token (which may represent a complex scientific data collection) in the input node, terminate with a single token in the output node (which represents the output data collection). In particular they always process all of the input data, leave no "debris data" behind and the output is always eventually computed.

1 Introduction

In this paper we are concerned with the creation of a formal language to define *dataflows*. Dataflows are often met in practice, examples are: *in silico* experiments of bioinformatics, systems processing data collected in physics, astronomy or other sciences. Their common feature is that large amounts of structured data are analyzed by a software system organized into a kind of network through which the data flows and is processed. The network consists of many servers, connected by communication protocols, i.e., HTTP or SOAP. In some cases advanced synchronization of the processing tasks is needed.

There are well-developed formalisms for *workflows* that are based on Petri nets [1]. However, we claim that for dataflows these should be extended with data manipulation aspects to describe workflows that manipulate structured complex values and where the structure of this data is reflected in the structure of the workflow. For this purpose we adopt the data model from the *nested relational calculus* which is a well-known and well-understood formalism in the domain of database theory.

Consequently, in a dataflow, tokens (which are generally assumed to be atomic in workflows) are typed and transport complex data values. Therefore, apart from classical places and transitions, we need transitions which perform operations on such data values. Of course, the operations are those of the nested relational calculus.

This way, the title equation emerges:

$$\text{Petri net} + \text{nested relational calculus} = \text{dataflow}.$$

The resulting language can be given a graphical syntax, thus allowing one to draw rather than write programs in it. This seems very important for a language whose programmers would not be professional computer scientists.

Next, we can give a formal semantics for dataflows. On the one hand, it is crucial, since we believe that formal, and yet executable, descriptions of all computational processes in the sciences should be published along with their domain-specific conclusions. Used for that purpose, dataflows can be precisely analyzed and understood, which is crucial for: (i) debugging by the authors, (ii) effective and objective assessment of their merit by the publication reviewers, and (iii) easy understanding by the readers, once published.

Next, the formal semantics makes it possible to perform formal analysis of the behavior of programs, including (automated) optimization and proving correctness.

We demonstrate the potential of the formal methods by proving the following (presented in an informal manner here).

Theorem. *Dataflows constructed hierarchically, i.e., according to a certain set of refinement rules we propose, are sound: initiated with a single token (which may represent a complex data value) in the input node, terminate with a single token in the output node (which represents the output value). In particular they always process all of the input data, leave no "debris data" behind and always terminate without a deadlock.*

We would like to point out that the above theorem is quite general—it applies uniformly to a very wide class of dataflows. Yet not every meaningful dataflow can be constructed hierarchically. However, we believe that the prevailing majority of those met in practice are indeed hierarchical.

Our idea of extending classical Petri nets is not new in general. Colored Petri nets permit tokens to be colored (with finitely many colors), and thus tokens carry some information. In the nets-within-nets paradigm [2] individual tokens have Petri net structure themselves. This way they can represent objects with their own, proper dynamics. Finally, self-modifying nets [3] assume standard tokens, but permit the transitions to consume and produce them in quantities functionally dependent on the occupancies of the places.

To compare, our approach assumes tokens to represent complex data values, which are however static. Only transitions are allowed to perform operations over the tokens' contents. Among them, the unnest/nest pairs act in such a way that the unnest transforms a single token with a set value into a set of tokens, and then the nest transforms the set of tokens back into a single "composite" token.

Also the introduction of complex value manipulation into Petri nets was already proposed in earlier work such as [4]. In this paper a formalism called NR/T-nets is proposed where places represent nested relations in a database schema and transitions represent operations that can be applied on the database. Although somewhat similar, the purpose of this formalism, i.e., representing the database schema and possible operations on it, is very different from the one presented here. For example, the structure of the Petri net in NR/T-nets does not reflect the workflow but only which relations are involved in which operations. Another example is the fact that in Dataflow nets we can easily integrate external functions and tools as special transitions and use them at arbitrary levels of the data structures. The latter is an important feature for describing and managing dataflows as found in scientific settings. Therefore, we claim that, together with other differences, this makes Dataflow nets a better formalism for representing dataflows.

1.1 What Is the Nested Relational Calculus?

The nested relational calculus (NRC) is a query language allowing one to describe functional programs using collection types, e.g. lists, bags, sets, etc. The most important feature of the language is the possibility to iterate over a collection. The only collections used in the following are sets, hence in the description below we ignore other collection types of NRC. NRC contains a set of base types. Moreover, it is allowed to combine these types to form records and sets.

Besides standard constructs enabling manipulation of records and set, NRC contains three constructs **sng**, **map** and **flatten**. For a value v of a certain type, **sng**(v) yields a singleton list containing v . Operation **map**, applied to a function from type τ to σ , yields a function from sets of type τ to sets of type σ .

Finally, the operation **flatten**, given a set of sets of type τ , yields a flattened set of type τ , by taking the union. These three basic operations are powerful enough for specifying functions by structural recursion over collections, cf. [5].

The inspiration for these constructs comes from the category-theoretical notion of a *monad* and the monadic approach to uniformly describe different notions of computation [6].

Under its usual semantics the NRC can already be seen as a dataflow description language but it only describes which computations have to be performed and not in what order, i.e., it is rather weak in expressing control flow. For some dataflows this order can be important because a dataflow can include calls to external functions and services which may have side-effects or are restricted by a certain protocol.

1.2 What are Petri Nets?

A classical Petri net is a bipartite graph with two types of nodes called the *places* and the *transitions*. The nodes are connected by directed edges. Only nodes of different kinds can be connected. Places are represented by circles and transitions by rectangles.

Definition 1 (Petri net). A Petri net is a triple $\langle P, T, E \rangle$ where:

- P is a finite set of places,
- T is a finite set of transitions ($P \cap T = \emptyset$),
- $E \subseteq (P \times T) \cup (T \times P)$ is a set of edges

A place p is called an *input place* of a transition t , if there exists an edge from p to t . Place p is called an *output place* of transition t , if there exists an edge from t to p . Given a Petri net $\langle P, T, E \rangle$ we will use the following notations:

$$\begin{array}{ll} \bullet p = \{t \mid \langle t, p \rangle \in E\} & p\bullet = \{t \mid \langle p, t \rangle \in E\} \\ \bullet t = \{p \mid \langle p, t \rangle \in E\} & t\bullet = \{p \mid \langle t, p \rangle \in E\} \\ \circ p = \{\langle t, p \rangle \mid \langle t, p \rangle \in E\} & p\circ = \{\langle p, t \rangle \mid \langle p, t \rangle \in E\} \\ \circ t = \{\langle p, t \rangle \mid \langle p, t \rangle \in E\} & t\circ = \{\langle t, p \rangle \mid \langle t, p \rangle \in E\} \end{array}$$

and their generalizations for sets:

$$\begin{array}{ll} \bullet A = \bigcup_{x \in A} \bullet x & A\bullet = \bigcup_{x \in A} x\bullet \\ \circ A = \bigcup_{x \in A} \circ x & A\circ = \bigcup_{x \in A} x\circ \end{array}$$

where $A \subseteq P \cup T$. Places are stores for *tokens*, which are depicted as black dots inside the places when describing the run a Petri net. Edges define the possible token flow. The semantics of a Petri net is defined as a transition system. A *state* is a distribution of tokens over places. It is often referred to as a *marking* $M \in P \rightarrow \mathbb{N}$. The state of a net changes when a transitions *fires*. For a transition t to fire it has to be *enabled*, that is, each of its input places has to contain at least one token. If transition t fires, it *consumes* one token from each of the places in $\bullet t$ and produces one token in each of the places in $t\bullet$.

Petri nets are a well-founded process modeling technique. The interest in them is constantly growing for the last fifteen years. Many theoretical results are available. One of the better studied classes are *workflow nets* used in business process workflow management[1].

Definition 2 (strongly connected). A Petri net is strongly connected if and only if for every two nodes n_1 and n_2 there is a directed path leading from n_1 to n_2 .

Definition 3 (workflow net). A Petri net $PN = \langle P, T, E \rangle$ is a workflow net if and only if:

- (i) PN has two special places: source and sink. The source has no input edges, i.e., $\bullet \text{source} = \emptyset$, and the sink has no output edges, i.e., $\text{sink}\bullet = \emptyset$.
- (ii) If we add to PN a transition t^* and two edges $\langle \text{sink}, t^* \rangle$, $\langle t^*, \text{source} \rangle$, then the resulting Petri net is strongly connected.

1.3 How We Combine NRC and Petri Nets

In this paper we propose a formal, graphical workflow language for data-centric scientific workflows. Since we call the type of workflows that we consider dataflows, we call the proposed language a *Dataflow language*. From NRC we inherit the set of basic operators and the type system. This should make reusing of existing database theory results easy. Dataflows could for example undergo optimization process as database queries do. To deal with the synchronization issues arising from processing of the data by distributed services we will use a Petri-net based formalism which is a clear and simple graphical notation and has an abundance of correctness analysis results. We believe that these techniques can be reused and combined with known results from database theory for verifying the correctness of dataflows which can be described in the proposed language.

2 Dataflow Language

The language we propose is a combination of NRC and Petri nets. We label transitions with labels determining functions or NRC operators, and associate NRC values with the tokens. As is usual with workflows that are described by Petri net we mandate one special input place and one special output place. If there is external communication this is modeled by transitions that correspond to calls to external functions. We use edge labeling to define how values of consumed tokens map onto the parameters of operations represented by transitions. To express conditional behavior we propose edge annotations indicating a condition that the value associated with the token must satisfy, so it can be transferred through the annotated edge. We also introduce two special transitions, unnest and nest, to enable explicit iteration over values of a collection.

A dataflow will be defined by an acyclic workflow net, transition and edge labeling, and edge annotation. The underlying Petri net will be called a *dataflow net*.

Definition 4 (dataflow net). A workflow net $WFN = \langle P, T, E \rangle$ is a dataflow net if and only if it has no cycles.

With the presence of the NRC map operation acyclicity seems not to be a strong limitation in real life applications. It has also an advantage as termination is always guaranteed.

2.1 The Type System

The dataflows are strongly typed, which means here that each transition consumes and produces tokens with values of a well determined type. The type of the value of a token is called the *token type*. The type system is similar to that of NRC. We assume a finite but user-extensible set of basic types which might for example be given by:

$$b ::= \text{boolean} \mid \text{integer} \mid \text{string} \mid \text{XML} \mid \dots$$

where *boolean* contains the boolean values *true* and *false*, *integer* contains all integer numbers, *string* contains all strings and *XML* contains all well-formed XML documents. Although this set can be arbitrarily chosen we will require that it at least contains the *boolean* type. From these basic types we can build complex types as defined by:

$$\tau ::= b \mid \langle l_1 : \tau_1, \dots, l_n : \tau_n \rangle \mid \{\tau\}$$

The type $\langle l_1 : \tau_1, \dots, l_n : \tau_n \rangle$, where l_i are distinct labels, is the type of all records having exactly fields l_1, \dots, l_n of types τ_1, \dots, τ_n respectively (records with no fields are also included). Finally $\{\tau\}$ is the type of all finite sets of elements of type τ . For later use we define CT to be the set of all complex types and CV the set of all possible complex values (note that the set of basic types is a subset of the set of complex types).

NRC can be defined on other collection types such as lists or bags. They are also included in existing scientific workflow systems such as in Taverna [7] where for example lists are supported. However, after careful analysis of various use cases in bioinformatics and examples distributed with existing scientific workflow systems we have concluded that sets are sufficient. Moreover, if lists are needed they can be simulated with sets in which the position is indicated by a number, but sets cannot be simulated with lists without introducing some arbitrary order on the elements. This order unnecessarily complicates the definition of the semantics and, as is known from database research, may limit optimization possibilities.

2.2 Edge Labels

Dataflows are not only models used to reason about data-processing experiments but are meant to be executed and produce computation results. Distinguishing transition input edges has to be possible to know how the tokens map onto the operation arguments. This is solved by edge labeling. Only edges leading from places to transitions are labeled. This labeling is determined by the edge naming function $EN : \circ T \rightarrow EL$ (note that $\circ T = P\circ$), where EL is some countably infinite set of edge label names, e.g., all strings over a certain non-empty alphabet.

2.3 Transition Labels

To specify desired operations and functions we label Petri net transitions. The transition labeling is defined by a transition naming function $TN : T \rightarrow TL$, where TL is a set of transition labels. Each transition label determines the number and labeling of input edges as well as the types of tokens that the transition consumes and produces when it fires. For this purpose the input typing and output typing functions are used: $IT : TL \rightarrow CT$ maps each transition label to the input type which must be a tuple type and $OT : TL \rightarrow CT$ maps each transition label to the output type.

2.4 Edge Annotation

To introduce conditional behavior we annotate edges with conditions. If an edge is annotated, then it can only transport tokens satisfying the condition. Conditions are visualized on diagrams in an UML [8] way, i.e., in square brackets. Only edges leading from places to transitions are annotated. There are four possible annotations defined by the edge annotation function:

$$EA : \circ T \rightarrow \{“=true”, “=false”, “=\emptyset”, “\neq\emptyset”, \varepsilon\}$$

The ε represents annotation absence. The meaning of the rest of the labels is self explanatory. For detailed specification see section 4.

2.5 Place Types

With each place in the dataflow net we associate a specific type that restricts the values that tokens in this place may have. This is represented by a function $PT : P \rightarrow CT$.

2.6 Dataflow

The dataflow net with edge naming, transition naming, edge annotation and place typing functions specifies a dataflow.

Definition 5 (dataflow). *Dataflow is a five-tuple $\langle DFN, EN, TN, EA, PT \rangle$ where:*

- $DFN = \langle P, T, E \rangle$ is a dataflow net,
- $EN : \circ T \rightarrow EL$ is an edge naming function,
- $TN : T \rightarrow TL$ is a transition naming function,
- $EA : \circ T \rightarrow \{“=true”, “=false”, “=\emptyset”, “\neq\emptyset”, \varepsilon\}$ is an edge annotation function,
- $PT : P \rightarrow CT$ is a place type function.

Since it is not true that in all the dataflows the types of the places are those that are required by the transitions we introduce the notion of *well-typed* dataflows. Informally, a dataflow is well-typed if for each transition (1) the names and types of its input places define a tuple type and it is the input tuple type of the transition, (2) the types of the output places are equal to the output type of the transition and (3) if any of the transitions input edges are annotated then the annotations match the types of the associated input places.

Definition 6 (well typed). *A dataflow $\langle DFN, EN, TN, EA, PT \rangle$ is well-typed if and only if for each transition $t \in T$ it holds that:*

1. *If $\circ t = \{(p_1, t), \dots, (p_n, t)\}$ and for all $1 \leq i \leq n$ it holds that $l_i = EN((p_i, t))$ and $\tau_i = PT(p_i)$ then $IT(TN(t)) = \langle l_1 : \tau_1, \dots, l_n : \tau_n \rangle$.*
2. *For each $(t, p) \in \circ t$ it holds that $PT(p) = OT(TN(t))$.*
3. *For each $(p, t) \in \circ t$ it holds that:*

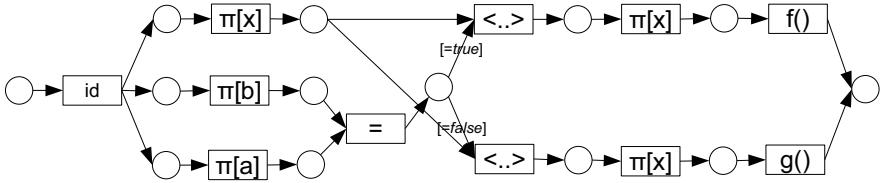


Fig. 1. If-then-else example

- if $EA((p, t)) \in \{\text{``=true''}, \text{``=false''}\}$ then $PT(p) = \text{boolean}$, and
- if $EA((p, t)) \in \{\text{``=\emptyset''}, \text{``\neq \emptyset''}\}$ then $PT(p)$ is a set type.

An example dataflow evaluating an **if** $u = v$ **then** $f(x)$ **else** $g(x)$ expression is shown on Fig. 1. Although the transition labels and a precise execution semantics are defined in the subsequent two sections, the example is self explanatory. First three copies of the input tuple of type $\langle u : b, v : b, x : \tau \rangle$ are made. Then each copy is projected to another field, a and b are compared, and a choice of upper or lower dataflow branch is made on the base of the boolean comparison result. The boolean value is disposed in a projection and depending on the branch that was chosen either $f(x)$ or $g(x)$ is computed.

3 Components

Since it is hard to keep track of new scientific analysis tools and data repositories, the language defines only a *core label subset*. Similarly to NRC the dataflow language can be extended with new *extension transition functions*. Such extension functions will usually represent computations done by external services. Examples from the domain of bioinformatics include: sequence similarity searches with BLAST [9], queries run on the Swiss-Prot [10] protein knowledgebase, or local enactments of the tools from the EMBOSS [11] package.

3.1 Core Transition Labels

The core transition labels are based on the NRC operator set plus two special *unnest* and *nest* labels, as shown in Table 1. A transition label is defined as a combination of the basic symbol (the first column) and a list of parameters which consists of types and edge labels (the second column). The values of the input type function IT and the output type function OT are given by the last two columns. For example, a concrete instance (i.e., with concrete parameter values) of the tuple constructor label would be $tl' = \langle \cdot \rangle_{a, \text{bool}, b, \text{int}}$ where the parameters are indicated in subscript and for which the functions IT and OT are defined such that $IT(tl') = OT(tl') = \langle a : \text{bool}, b : \text{int} \rangle$. Another example would be $tl'' = \pi[a]_{a, \text{bool}, b, \text{int}}$ where $IT(tl'') = \langle a : \text{bool}, b : \text{int} \rangle$ and $OT(tl'') = \text{bool}$. The remaining core transition labels are defined in Table 1 in a similar fashion.

Table 1. Core transition labels

Sym.	Parameters	Operation name	Input type	Output type
\emptyset	ι, τ_1, τ_2	empty-set constr.	$\langle l : \tau_1 \rangle$	$\{\tau_2\}$
$\{\cdot\}$	ι, τ	singleton-set constr.	$\langle l : \tau \rangle$	$\{\tau\}$
\cup	ι_1, ι_2, τ	set union	$\langle l_1 : \{\tau\}, l_2 : \{\tau\} \rangle$	$\{\tau\}$
φ	ι, τ	flatten	$\langle l : \{\{\tau\}\} \rangle$	$\{\tau\}$
\times	$\iota_1, \tau_1, \iota_2, \tau_2$	Cartesian product	$\langle l_1 : \{\tau_1\}, l_2 : \{\tau_2\} \rangle$	$\{\langle l_1 : \tau_1, l_2 : \tau_2 \rangle\}$
$=$	ι_1, ι_2, b	atomic-value equal.	$\langle l_1 : b, l_2 : b \rangle$	<i>boolean</i>
$\langle \rangle$	ι, τ	empty tuple constr.	$\langle l : \tau \rangle$	$\langle \rangle$
$\langle \cdot \rangle$	$\iota_1, \tau_1, \dots, \iota_n, \tau_n$	tuple constr.	$\langle l_1 : \tau_1, \dots, l_n : \tau_n \rangle$	$\langle l_1 : \tau_1, \dots, l_n : \tau_n \rangle$
$\pi[l_i]$	$\iota, \langle \iota_1 : \tau_1, \dots, \iota_n : \tau_n \rangle$	field projection	$\langle l : \langle l_1 : \tau_1, \dots, l_n : \tau_n \rangle \rangle$	τ_i
id	ι, τ	identity	$\langle l : \tau \rangle$	τ
$*$	ι, τ	unnest	$\langle l : \{\tau\} \rangle$	τ
$*^{-1}$	ι, τ	nest	$\langle l : \tau \rangle$	$\{\tau\}$

3.2 Extension Transition Labels

Next to the set of core transition labels the set of transition labels TL also consists of user-defined transition labels. As for all transition labels the functions IT and OT must be defined for each of them. Moreover, for every user-defined transition label tl we will assume that there exists an associated function $\Phi_{tl} : IT(tl) \rightarrow OT(tl)$ which represents a possibly non-deterministic computational function that is performed when the transition fires.

To give a concrete example a bioinformatician may define a $getSWPrByAC$, for which $IT(getSWPrByAC) = \langle ac : string \rangle$ and $OT(getSWPrByAC) = XML$. The $\Phi_{getSWPrByAC}$ function would represent a call to a Swiss-Prot knowledgebase and return a XML formated entry for a given primary accession number.

4 Transition System Semantics

Let the $\langle DFN, EN, TN, EA, PT \rangle$ be a well-typed dataflow (if not stated otherwise this will be assumed in the rest of the paper). Its semantics is given as a transition system (see subsection 4.2). Each place contains zero or more *tokens*, which represent data values. Formally a token is a pair $k = \langle v, h \rangle$, where $v \in CV$ is the transported value and $h \in H$ is this value's *unnesting history*. This unnesting history is defined in the next subsection (see 4.1). The set of all possible tokens is then $K = CV \times H$. By the type of a token we mean the type of its value, i.e., $\langle v, h \rangle : \tau$ if and only if $v : \tau$.

The state of a dataflow, also called *marking*, is the distribution of tokens over places $M \in (P \times K) \rightarrow \mathbb{N} \cup \{0\}$ where $M(p, k) = n$ means that place p contains n times the token k . We only consider as states distributions for which token types match types of places they are in, i.e., for all places $p \in P$ and tokens $k \in K$ such that $M(p, k) > 0$ it holds that $k : PT(p)$. Transitions are the active components

in a dataflow: they can change the state by *firing*, that is consuming tokens from each of their input places and producing tokens in each of their output places. In distinction to workflow nets, for some transitions (nests and unnests) more than one token per input place can be consumed and an arbitrary number of tokens per output place can be produced. A transition that can fire in a given state is called *enabled*. The types, numbers and unnesting history conditions of tokens in input places for a given transition to be enabled are determined by its transition label.

We adopt the following Petri net notations:

- $M_1 \xrightarrow{t} M_2$: the transition t is enabled in state M_1 and firing t in M_1 results in state M_2
- $M_1 \rightarrow M_2$: there is a transition t such that $M_1 \xrightarrow{t} M_2$
- $M_1 \xrightarrow{\theta} M_n$: the firing sequence $\theta = t_1 t_2 \dots t_{n-1}$ leads from state M_1 to state M_n , i.e., $\exists_{M_2, M_3, \dots, M_{n-1}} M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} M_3 \xrightarrow{t_3} \dots \xrightarrow{t_{n-1}} M_n$
- $M_1 \xrightarrow{*} M_n$: $M_1 = M_n$ or there is a firing sequence $\theta = t_1 t_2 \dots t_{n-1}$ such that $M_1 \xrightarrow{\theta} M_n$

A state M_n is called *reachable* from M_1 if and only if $M_1 \xrightarrow{*} M_n$.

Although the semantics of a dataflow is presented as a transition system, as in classical Petri nets, two enabled transitions may fire concurrently, if there is enough input tokens.

4.1 Token Unnesting History

Every time it fires, an unnest transition (see 4.2) consumes one token with a set value and produces a token for each element in this set. The information about the unnested set and the particular element of that set for which a given token was created is stored in that token's unnesting history. This is illustrated in Fig. 2 where in (a) we see in the first place a single token with value $\{1, 2, 3\}$ and an empty history $()$. When the unnest transition fires it produces a token for each element as shown in (b). The history is then extended with a pair that contains (1) the set that was unnested and (2) the element for which this particular token was produced. As shown in (c) normal transitions will produce tokens with histories identical to that of the consumed input tokens. Once all the tokens that belong to the same unnesting group have arrived in the input place of the nest transition as is shown in (d), which can be verified by looking at their histories, then the nest transition can fire and combine them into a single token as is shown in (e). Note that where the unnest transition adds a pair to the history, the nest transition removes a pair from the history. Since sets can be unnested and nested several times, the history is a *sequence* of pairs where each pair contains the unnesting information of one unnesting step. Therefore we formally define the *set of all histories H* as the set of all sequences of pairs $\langle s, x \rangle$, where $s \in CV$ is a set and $x \in s$.

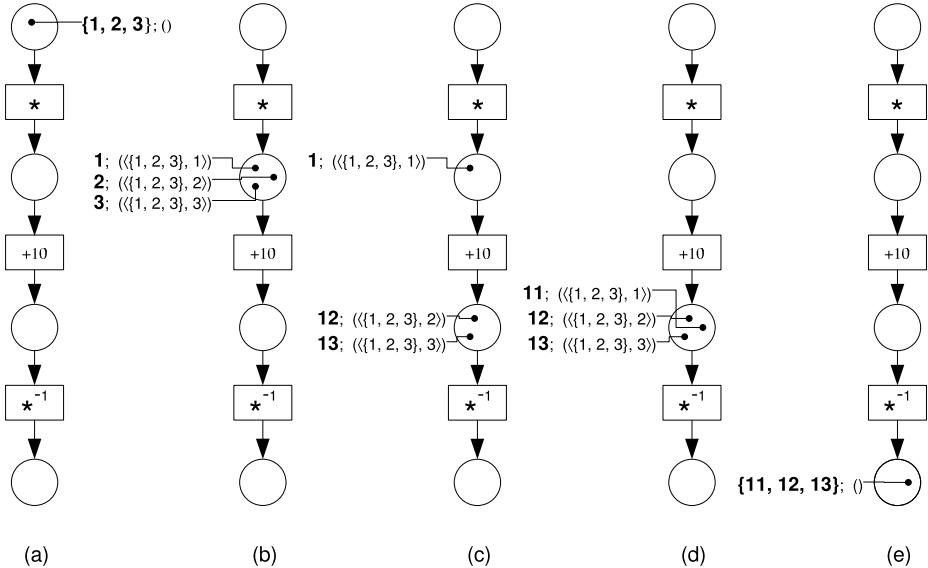


Fig. 2. An illustration of the unnesting history of tokens

4.2 Semantics of Transitions

The following shortcut will be used, since tokens can only flow along a condition-annotated edge if the value of the token satisfies the condition:

$$\langle v, h \rangle \curvearrowright e \stackrel{\text{def}}{=} (EA(e) = \varepsilon \Rightarrow \mathbf{true}) \wedge \\
 (EA(e) = "\mathbf{=true}" \Rightarrow v = \mathbf{true}) \wedge \\
 (EA(e) = "\mathbf{=false}" \Rightarrow v = \mathbf{false}) \wedge \\
 (EA(e) = "\mathbf{=\emptyset}" \Rightarrow v = \emptyset) \wedge \\
 (EA(e) = "\mathbf{\neq\emptyset}" \Rightarrow v \neq \emptyset)$$

A formal definition for unnest, nest and extension transition labels will be given. The nest and unnest are special since only transitions labeled in this way change the token's unnesting history. Moreover nest transitions can consume more than one token per input place and unnest transitions can produce more than one token per output place. Transitions labeled by other labels behave as in classical Petri nets except that each time they fire all consumed tokens must have identical histories. The semantics of the rest of the core transition labels fully agrees with the intuitions given by their names. In particular, they consume and produce exactly one token when firing, do not change the unnesting history and the formulas are analogous to those given for extension transition labels.

Unnest. For an unnest transition $t \in T$ it holds that $M_1 \xrightarrow{t} M_2$ if and only if there exists a token $\langle v, h \rangle \in K$ such that:

1. for all places $p \in \bullet t$ it holds that:
 - (a) $\langle v, h \rangle \curvearrowright \langle p, t \rangle$,
 - (b) $M_2(p, \langle v, h \rangle) = M_1(p, \langle v, h \rangle) - 1$ and
 - (c) $M_2(p, \langle v', h' \rangle) = M_1(p, \langle v', h' \rangle)$ for all tokens $\langle v', h' \rangle \neq \langle v, h \rangle$
2. for all places $p \in t \bullet$ it holds that:
 - (a) $M_2(p, \langle x, h \oplus \langle v, x \rangle \rangle) = M_1(x, \langle v, h \oplus \langle v, x \rangle \rangle) + 1$ for every $x \in v$ and
 - (b) $M_2(p, \langle v', h' \rangle) = M_1(p, \langle v', h' \rangle)$ if $\langle v', h' \rangle \neq \langle x, h \oplus \langle v, x \rangle \rangle$ for all $x \in v$
3. for all places $p \notin \bullet t \cup t \bullet$ it holds that $M_2(p, \langle v', h' \rangle) = M_1(p, \langle v', h' \rangle)$ for all tokens $\langle v', h' \rangle \in K$

where $(a_1, a_2, \dots, a_n) \oplus a_{n+1} := (a_1, a_2, \dots, a_n, a_{n+1})$.

Nest. For a nest transition $t \in T$ it holds that $M_1 \xrightarrow{t} M_2$ if and only if there exists a history h_S , a set $s = \{x_1, \dots, x_n\} \in CV$ and a set of tokens $S = \{\langle v_1, h_S \oplus \langle s, x_1 \rangle \rangle, \dots, \langle v_n, h_S \oplus \langle s, x_n \rangle \rangle\} \subseteq K$ such that

1. for all places $p \in \bullet t$ it holds that:
 - (a) $\langle v_i, h_i \rangle \curvearrowright \langle p, t \rangle$ for each $\langle v_i, h_i \rangle \in S$,
 - (b) $M_2(p, \langle v_i, h_i \rangle) = M_1(p, \langle v_i, h_i \rangle) - 1$ for each $\langle v_i, h_i \rangle \in S$,
 - (c) $M_2(p, \langle v', h' \rangle) = M_1(p, \langle v', h' \rangle)$ for each $\langle v', h' \rangle \notin S$
2. for all places $p \in t \bullet$ and assuming that $v_S = \{v_1, \dots, v_n\}$ it holds that:
 - (a) $M_2(p, \langle v_S, h_S \rangle) = M_1(p, \langle v_S, h_S \rangle) + 1$ and
 - (b) $M_2(p, \langle v', h' \rangle) = M_1(p, \langle v', h' \rangle)$ for all tokens $\langle v', h' \rangle \neq \langle v_S, h_S \rangle$
3. for all places $p \notin \bullet t \cup t \bullet$ it holds that $M_2(p, \langle v', h' \rangle) = M_1(p, \langle v', h' \rangle)$ for all tokens $\langle v', h' \rangle \in K$

where $(a_1, a_2, \dots, a_n) \oplus a_{n+1} := (a_1, a_2, \dots, a_n, a_{n+1})$.

Extensions. For an extension transition $t \in T$ that is labeled with an extension transition label it holds that $M_1 \xrightarrow{t} M_2$ if and only if there exists a history h such that with each place $p \in \bullet t$ we can associate a value $v_p \in CV$ such that

1. for all places $p \in \bullet t$ it holds that
 - (a) $\langle v_p, h \rangle \curvearrowright \langle p, t \rangle$,
 - (b) $M_2(p, \langle v_p, h \rangle) = M_1(p, \langle v_p, h \rangle) - 1$ and
 - (c) $M_2(p, \langle v', h' \rangle) = M_1(p, \langle v', h' \rangle)$ for all $\langle v', h' \rangle \neq \langle v_p, h \rangle$
2. for all places $p \in t \bullet$ it holds that if $v = \Phi_{TL(t)}(\langle l_1 : v_1, \dots, l_n : v_n \rangle)$ where $\{\langle l_1, v_1 \rangle, \dots, \langle l_n, v_n \rangle\} = \{\langle EN(p, t), v_p \rangle \mid p \in \bullet t\}$ then
 - (a) $M_2(p, \langle v, h \rangle) = M_1(p, \langle v, h \rangle) + 1$ and
 - (b) $M_2(p, \langle v', h' \rangle) = M_1(p, \langle v', h' \rangle)$ if $\langle v', h' \rangle \neq \langle v, h \rangle$
3. for all places $p \notin \bullet t \cup t \bullet$ it holds that $M_2(p, \langle v', h' \rangle) = M_1(p, \langle v', h' \rangle)$ for all tokens $\langle v', h' \rangle \in K$

5 Hierarchical Dataflows

In spite of the fact that we extend workflow Petri nets, existing technical and theoretical results can be easily reused. This is what we intend to demonstrate here.

The dataflow language is developed to model data-centric workflows and in particular scientific data processing experiments. The data to be processed should be placed in the dataflow's source and after the processing ends the result should appear in its sink. A special notation is introduced that distinguishes two state families. The state $input_k$ is an input state with a single token k in the source place and all other places empty, that is $input_k(\langle source, k \rangle) = 1$ and $input_k(\langle p, k' \rangle) = 0$ if $k \neq k'$ or $p \neq source$. Similarly $output_k$ is an output state with single token in the sink place. Starting with one token in the source place and executing the dataflow may not always produce a computation result in the form of a token in the sink place. For some dataflows the computation may halt in a state in which none of the transitions is enabled yet the sink is empty. Even reaching a state in which there are no tokens at all is possible. Examples of such incorrect dataflows are shown on Fig. 3.

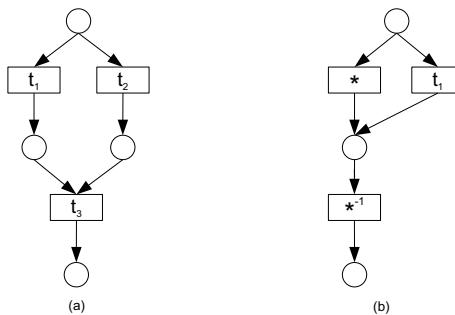


Fig. 3. Incorrect dataflows

For the dataflow (a) the token from the source can be consumed by a transition t_1 or t_2 , but not by both of them at the same time. Transition t_3 will not become enabled then, because one of its input places will stay empty. In the (b) case, if t_1 gets the source token, the $*^{-1}$ does not become enabled, because only $*$ can produce a token with the required unnesting history. But it may even be not enough when the $*$ transition consumes the source token. If the source token carried an empty set, then in the resulting state all places would be empty.

Similar incorrectness was also studied in context of procedures modeled by classical workflow nets. The correct procedures are called *sound* [1]. The notion of soundness can in a natural way be adapted to dataflows:

Definition 7 (soundness). A dataflow $\langle DFN, EN, TN, EA, PT \rangle$, with $sink : \tau$ and $source : \theta$, is sound if and only if for each token $k' : \tau$ there exists token $k'' : \theta$ that:

- (i) $\forall_M (\text{input}_{k'} \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} \text{output}_{k''})$
- (ii) $\forall_M (\text{input}_{k'} \xrightarrow{*} M \wedge M(\text{out}, k'') > 0) \Rightarrow (M = \text{output}_{k''})$
- (iii) $\forall_{t \in T} \exists_{M, M'} \text{input}_{k'} \xrightarrow{*} M \xrightarrow{t} M'$

5.1 Refinement Rules

To avoid designing of unsound dataflows we propose a structured approach. In a *blank dataflow generation step* a pattern dataflow is constructed in a top-down manner, starting from a single place and performing refinements using a given set of rules. Each refinement replaces a place or transition with larger subnet. In the generated *blank dataflow* all transitions are unlabeled, but for each we indicate if it will or will not be labeled with a nest and unnest.

The seven basic refinement rules are shown on Fig. 4. Each rule, except the third one, can be applied only if the transformed node has both input and output edges. All the input edges of the transformed node are copied to all the resulting entry nodes, and an analogous rule applies to the output edges. The rules and the aim to make dataflows structured as in structured programming languages were motivated by the work done on workflow nets by Piotr Chrząstowski [12].

Definition 8 (blank dataflow). A *blank dataflow* is a tuple $\langle DFN, NP, EA \rangle$ where:

- $DFN = \langle P, T, E \rangle$ is a dataflow net,
- $NP : T \rightarrow \{*, *^{-1}, \text{blank}\}$ is a nesting plan function,
- $EA : E \rightarrow \{\text{``=true''}, \text{``=false''}, \text{``}=\emptyset\text{''}, \text{``}\neq\emptyset\text{''}, \varepsilon\}$ is an edge annotation function.

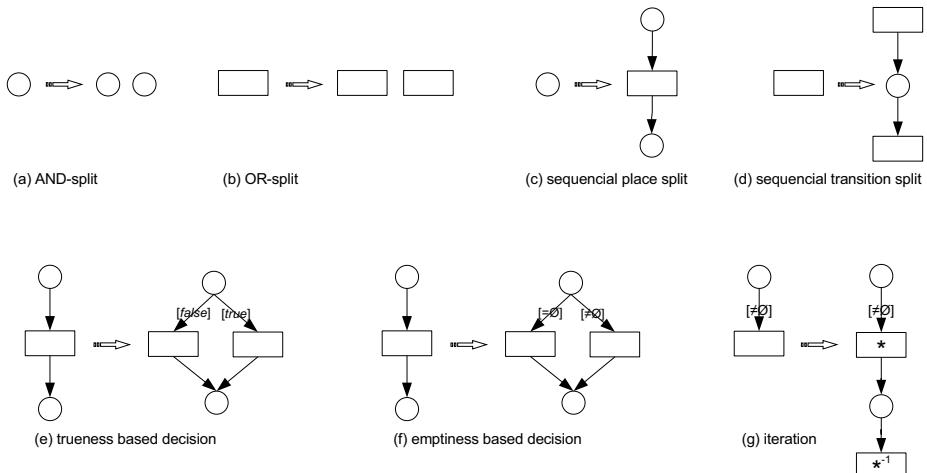


Fig. 4. Refinement rules

Definition 9 (blank dataflow generations step). We start with a blank dataflow $\langle DFN, NP, EA \rangle$ consisting of a single place with no transitions and perform the transformations presented on Fig. 4 with the constraint that, except for the sequential place split all the transformations may be applied only if a node has at least one input and at least one output.

Definition 10 (hierarchical dataflow). Let $DF = \langle DFN, EN, TN, EA, PT \rangle$ be a well-typed dataflow obtained by labeling of all transitions in a blank dataflow net $BDF = \langle DFN, NP, EA \rangle$ under following conditions:

- (i) a transition is labeled as nesting if and only if it was planned to, that is:
 $\exists_{c \in CT} TN(t) = nest_c$ if and only if $\forall_{t \in T} NP(t) = *$
- (ii) a transition is labeled as unnesting if and only if it was planned to, that is:
 $\exists_{c \in CT} TN(t) = nest_c$ if and only if $\forall_{t \in T} NP(t) = *^{-1}$

The dataflow DF is hierarchical if and only if the blank dataflow net BDF can be constructed in a blank dataflow generation step.

Theorem 1. All hierarchical dataflows are sound.

Proof. (sketch) The proof follows the one given in [12]. It can easily be checked that making any of the refinement rules doesn't jeopardize the possibility of obtaining a sound dataflow by labeling of a blank dataflow. The rest of the proof proceeds by the induction on the number of refinements performed. \square

6 A Bioinformatics Dataflow Example

In [13] it was illustrated that NRC is expressive enough to describe real life dataflows in bioinformatics. In this work we combine NRC with Petri nets, using the more convenient Petri net notation for explicitly defining the control flow. In this section we also present a dataflow based on a real bioinformatics example [14]. The corresponding dataflow net is given in Fig. 5. The goal of this dataflow is to find differences in peptide content of two samples of cerebrospinal fluid (a peptide is an amino acid polymer). One sample belongs to a diseased person and the other to a healthy one. A mass spectrometry wet-lab experiment has provided data about observed polymers in each sample. A peptide-identification algorithm was invoked to identify the sequences of those polymers, providing an amino-acid sequence and a confidence score for each identified polymer. The dataflow starts with a tuple containing two sets of data from the identification algorithm one obtained from the “healthy” sample and the other from the “diseased” sample: complex input type $\langle \text{healthy} : PepList, \text{diseased} : PepList \rangle$ with complex type $PepList = \{\langle \text{peptide} : \text{String}, \text{score} : \text{Number} \rangle\}$. Each data set contains tuples consisting of an identified peptide, represented by base type **String**, and the associated confidence score, represented by base type **Number**. The dataflow transforms this input into a set of tuples containing the identified peptide, a singleton containing the confidence score from the “healthy” data set or empty set if the identified peptide was absent in the “healthy” data set, and similarly, the confidence score from the “diseased” data set. The complex output type is the following: $\{\langle \text{peptide} : \text{String}, \text{healthy} : \{\text{Number}\}, \text{diseased} : \{\text{Number}\} \rangle\}$.

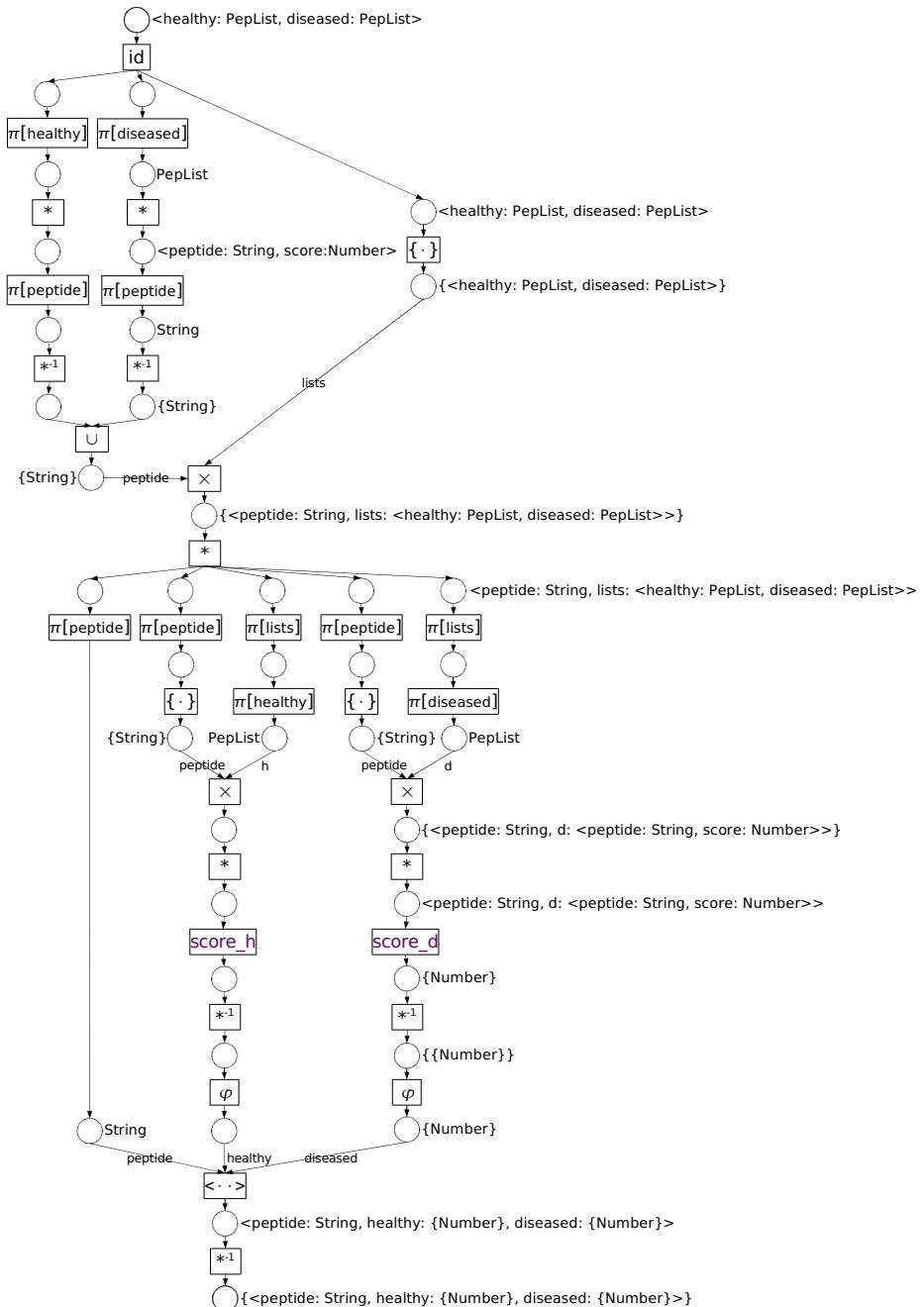


Fig. 5. Finding differences in peptide content of two samples

7 Conclusions and Further Research

In this paper we have presented a graphical language for describing dataflows, i.e., workflows where large amounts of complex data are manipulated and the structure of the manipulated data is reflected in the structure of the workflow. In order to be able to describe both the control flow and the data flow the language is based on Petri nets and the nested relational calculus (NRC) and has a formal semantics that is based upon these two formalisms. This ensures that from the large body of existing research on these we can reuse or adapt certain results. This is illustrated by taking a well-known technique for generating sound workflow nets and using it to generate sound dataflow nets. We have shown that the technique that restricts itself to hierarchical dataflows and the technique that goes beyond, can both be readily applied to our formalism.

In future research we intend to compare, investigate and extend this formalism in several ways. Since the dataflow nets tend to become quite large for relatively simple dataflows, we intend to introduce more syntactic sugar. We also want to investigate whether a similar control-flow semantics can be given for the textual NRC and see how the two formalisms compare under these semantics. Since existing systems for data-intensive workflows often lack formal semantics, we will investigate if our formalism can be used to provide these. It is also our intention to add the notions of *provenance* and *history* to the semantics such that these can be queried with a suitable query language such as the NRC. This can be achieved in a straightforward and intuitive way by remembering all tokens that passed through a certain place and defining the provenance as a special binary relation over these tokens. Storing all these tokens makes it not only possible to query the history of a net but also to reuse intermediate results of previous versions of a dataflow. Another subject is querying dataflows where a special language is defined to query dataflow repositories to, for example, find similar dataflows or dataflows that can be reused for the current research problem. Since dataflow nets are essentially labeled graphs it seems likely that a suitable existing graph-based query formalism could be found for this. Finally we will investigate the possibilities of workflow optimization by applying known techniques from NRC research. Since optimization often depends on the changing of the order of certain operations it will then be important to extend the formalism with a notion of “color” for extension transitions that indicates whether their relative order may be changed by the optimizer.

References

1. van der Aalst W.: The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers* (1998) 21–66
2. Valk, R.: Object Petri nets: Using the nets-within-nets paradigm. In: *Lectures on Concurrency and Petri Nets*. (2003) 819–848
3. Valk, R.: Self-modifying nets, a natural extension of Petri nets. In: *ICALP*. (1978) 464–476

4. Oberweis, A., Sander, P.: Information system behavior specification by high level petri nets. *ACM Trans. Inf. Syst.* **14** (1996) 380–420
5. Buneman, P., Naqvi, S., Tannen, V., Wong, L.: Principles of programming with complex objects and collection types. *Theoretical Computer Science* (1995) 3–48
6. Moggi, E.: Notions of computation and monads. *Information and Computation* (1991) 55–92
7. Oinn, T., Addis, M., Ferris, J., Marvin, D., Greenwood, M., Carver, T., Wipat, A., Li, P.: Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics* (2004)
8. Object Management Group: Unified modeling language resource page. <http://www.uml.org/>
9. Altschul, S., Gish, W., Miller, W., Myers, E., Lipman, D.: Basic local alignment search tool. *J. Mol. Biol.* (1990) 403–410
10. Boeckmann, B., Bairoch, A., Apweiler, R., Blatter, M., Estreicher, A., et al.: The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic Acids Research* **31** (2003) 365–370
11. Rice, P., Longden, I., Bleasby, A.: Emboss: The european molecular biology open software suite (2000). *Trends in Genetics* **16** (2000) 276–277
12. Chrząstowski-Wachtel, P., Benatallah, B., Hamadi, R., O'Dell, M., Susanto, A.: A top-down petri net-based approach for dynamic workflow modeling. In: *Proceedings of Business Process Management: International Conference, BPM 2003*. Volume 2678 of *Lecture Notes in Computer Science*, Springer (2003) 336–353
13. Gambin, A., Hidders, J., Kwasnikowska, N., Lasota, S., Sroka, J., Tyszkiewicz, J., Van den Bussche, J.: NRC as a formal model for expressing bioinformatics workflows. Poster at ISMB 2005 (2005)
14. Dumont, D., Noben, J., Raus, J., Stinissen, P., Robben, J.: Proteomic analysis of cerebrospinal fluid from multiple sclerosis patients. *Proteomics* **4** (2004)

On the Controlled Evolution of Access Rules in Cooperative Information Systems

Stefanie Rinderle^{1,*} and Manfred Reichert²

¹ Department Databases and Information Systems, University of Ulm, Germany
`rinderle@informatik.uni-ulm.de`

² Information Systems Group, University of Twente, The Netherlands
`m.u.reichert@cs.utwente.nl`

Abstract. For several reasons enterprises are frequently subject to organizational change. Respective adaptations may concern business processes, but also other components of an enterprise architecture. In particular, changes of organizational structures often become necessary.

The information about organizational entities and their relationships is maintained in organizational models. Therefore the quick and correct adaptation of these models is fundamental to adequately cope with changes. However, model changes alone are not sufficient to guarantee consistency. Since organizational models also provide the basis for defining access rules (e.g., actor assignments in workflow management systems or access rules in document-centered applications) this information has to be adapted accordingly (e.g., to avoid non-resolvable actor assignments). Current approaches do not adequately address this problem, which often leads to security gaps and delayed change adaptations.

In this paper we present a comprehensive approach for the controlled evolution of organizational models in cooperative information systems. First, we introduce a set of operators with well-defined semantics for defining and changing organizational models. Second, we present an advanced approach for the semi-automated adaptation of access rules when the underlying organizational model is changed. This includes a formal part concerning both the evolution of organizational models and the adaptation of related access rules.

1 Introduction

Enterprise-wide, cooperative information systems (IS) comprise a variety of application and system components. Important tasks to be accomplished include the support of business processes, the management of enterprise documents and the integration of enterprise applications. For the implementation of such services different middleware components exists, like workflow systems, document management systems, and tools for enterprise application integration [1, 2, 3].

The controlled access to its application services as well as to the application objects managed by them (e.g., business processes, business documents,

* This research work was conducted during a postdoc stay at the University of Twente

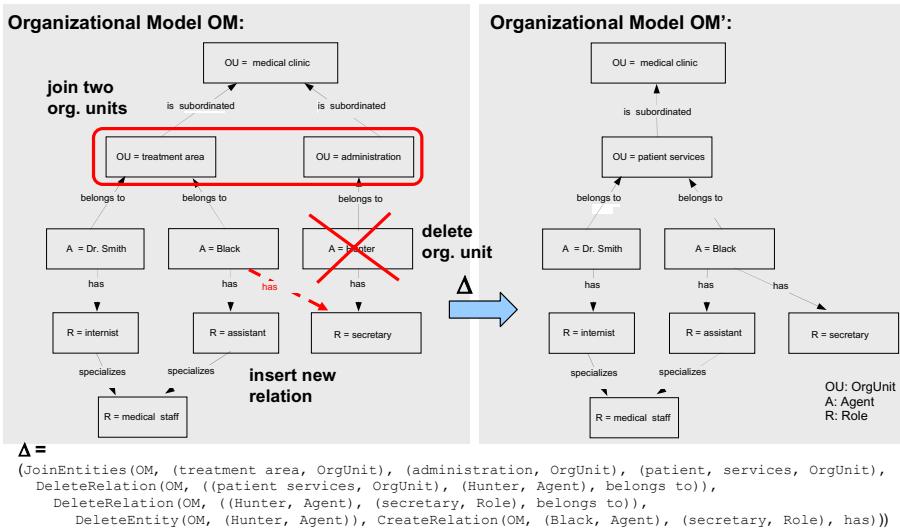


Fig. 1. Example of an Organizational Model and a Model Change (simplified)

resources, application systems, etc.) constitutes an important task for any cooperative IS. Usually, this results in a large number of access rules covering different system aspects and user privileges. Furthermore, these rules have to be frequently adapted due to changes of organizational structures [4, 5, 6]. Such organizational changes become necessary, for instance, when an organizational unit is split into two sub-units, two existing units are joined, a group of users is reassigned to a new unit, or simply an employee leaves the organization.¹ As a consequence, the access rules whose definition is based on organizational entities may have to be adapted as well. So far, the controlled evolution of access rules in cooperative IS has not been addressed in sufficient detail, which has led to severe security gaps when organizational changes are introduced.

Typically, information about organizational entities (e.g., organizational units, roles, and users) and the relations between them (e.g., assignment of a user to a role or an organizational unit) is kept in an organizational model. Based on such a model, access rights and user privileges can be defined (e.g., actor assignments in a workflow management system or access rules in document-centered applications). Consequently, when organizational changes occur, both the organizational model and the related access rules have to be adapted in a consistent manner.

Another (practical) problem arises from the fact that the (middleware) components used to build the application services of cooperative IS often maintain their own organizational model and security component; i.e., the information about organizational entities (and their relations) as well as the access rules

¹ For respective results from one of our case studies in the clinical domain see [4].

based on them may be scattered over different system components. On the one hand this has led to functional redundancy, on the other hand (heterogeneous) information about organizational structures is kept redundantly in different security components. The latter very often results in inconsistencies, high costs for system maintainability, and inflexibility when dealing with organizational change.

But even if the organizational model is kept in a central repository, the problem of maintainability remains. The correct and consistent adaptation of the organizational model is only one side of the coin when dealing with organizational changes; the other is to correctly and efficiently adapt the access rules defined on basis of this model. Note that in large environments hundreds up to thousands of access rules may exist, each of them capturing different privileges of the cooperative IS. This, in turn, makes it a hard job for the system administrator to quickly and correctly adapt these rules to changes of the organizational model(s).

Current approaches do not sufficiently deal with this issue. They neither make use of the semantics of the applied model changes nor do they provide any automated support for rule adaptation. In practice, this often leads to problems like non-resolvable actor assignments, unauthorized access to business documents, or inconsistent user worklists. Assume, for example, that two organizational units are joined in order to make the enterprise more efficient (cf. Fig. 1). If this change is performed in an uncontrolled manner, it may result in non-resolvable access rules referring to one of these two units (no longer present in the new organizational model).

Facing these challenges we need a logically centralized component which manages the organizational model and its changes in a consistent and efficient manner. Furthermore, model changes have to be propagated to access rules (used within different system components) in a correct and efficient manner. Finally, we have to consider both, access rules which are checked when a certain privilege is applied (e.g., when a user wants to access a document) and rules which are used to determine a set of authorized users (e.g., the set of actors who may work on a certain workflow activity).

In this paper we present a comprehensive approach for the controlled evolution of organizational models in cooperative IS. This approach complements our previous work on the controlled evolution of process models and process instances in adaptive process management systems [7, 8, 9, 10, 11]. First, we introduce a set of operations with well-defined semantics for defining and changing organizational models. Second, we present an advanced approach for the semi-automated adaptation of access rules when the referred organizational model is modified. For selected organizational changes we show how they can be realized in our formal framework and how their effects on access rules look like. We then try to derive migration strategies for affected access rules. Thereby we make use of the semantics of the applied model changes and we introduce formally sound migration concepts. For this purpose, we introduce a formal meta model for defining and changing organizational models and related ac-

cess rules. Altogether this paper includes a formal part concerning the evolution of organizational models and related access rules as well as a discussion of practical issues.

Section 2 discusses related work. In Section 3 we present a sample meta model for defining organizational models and access rules. Section 4 deals with the evolution of organizational models and the (semi-automated) adaptation of related access rules. Use cases and practical issues are sketched in Section 5. We conclude with a short summary and an outlook on future work.

2 Related Work

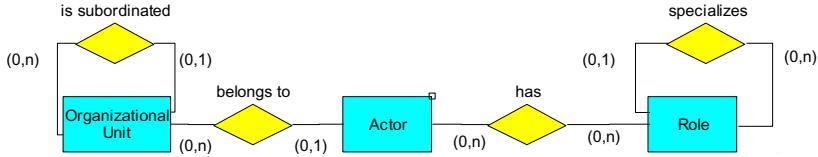
The provision of adequate access control mechanism is indispensable for any cooperative IS. In the literature many approaches exist dealing with corresponding issues (e.g., [6, 12, 13, 14]). Most of them use *Role-Based Access Control (RBAC)* models for defining and managing user privileges [15, 16, 12, 17], e.g., for ensuring the controlled access to business documents when using *document management* technology [18] or for resolving the set of actors that qualify for a certain task in a *workflow management system* [19, 20, 21, 13, 14]. Regarding workflow-based applications, in addition, dynamic constraints (e.g., separation of duties) have been considered [20, 21]. So far, however, only few approaches [22, 23] have addressed the problem of organizational change (see below).

Issues related to the modeling of organizational structures have been considered by different groups [5, 13, 24]. Most of them suggest a particular meta model for capturing organizational entities and their relations. Model changes and the adaptation of access rules, however, have not been studied by these approaches in sufficient detail.

In [25] several issues related to changes of process and organizational structures have been discussed. In this work the authors also motivate the need for the controlled change of organizational models. In particular, they discuss different kinds of adaptations that have to be supported (e.g., to extend, reduce, replace, and re-link model elements). However, no concrete solution approach is provided (like, for example, formal change operators with well-defined semantics or mechanisms for adapting access rules after model changes).

In [6, 26] the author identifies eight different categories for structural changes of organizational models. Examples of such change categories include the splitting of organizational units, the creation of new organizational entities, and the re-linkage of a user to a new unit. In principle, all these cases can be captured by our change framework as well. As opposed to [6], however, we have followed a rigorous formal approach in order to be able to derive the effects of organizational changes on related access rules as well. This issue has not been addressed in [6].

Other approaches have introduced role-based access control model for adaptive workflows [14, 23]. In [23] the authors additionally address issues related to the evolution of access rights in workflow management systems. However, no formal considerations are made and only simple cases are managed when compared to our approach.

**Fig. 2.** Organizational Meta Model

3 Organizational Models and Access Rules

In order to be able to reason about organizational changes and their concrete impact on access rules we need a formalization of organizational structures. For this purpose, first of all, we introduce an (organizational) meta model, which is comparable to existing RBAC models (e.g., [15]) and which can be used for describing organizational entities and the relations between them (cf. Fig. 2). Due to lack of space, in this paper we restrict our considerations to the basic entity types *organizational unit*, *role* and *actor*, and to the particular relation types existing between them (e.g., actor A_1 belongs to organizational unit O_1 , role R_1 specializes role R_0 , etc.). In our complete framework currently implemented in the ADEPT2 project, we also consider entity types like *position*, *group* or *capability* when defining and changing organizational models [24].

Regarding the meta model *OMM* assumed in this paper (cf. Fig. 2) we specify the set of valid entity types and the set of valid relation types as follows:

- $\text{EntityTypes} := \{\text{OrgUnit}, \text{Actor}, \text{Role}\}$
- $\text{RelationTypes} := \{(\text{OrgUnit}, \text{OrgUnit}, \text{is subordinated}), (\text{Role}, \text{Role}, \text{specializes}), (\text{Actor}, \text{OrgUnit}, \text{belongs to}), (\text{Actor}, \text{Role}, \text{has})\}$

We further denote

- $\mathcal{E} := \mathcal{E}_{Id} := \{(entId, entType) \mid entId \in Id, entType \in \text{EntityTypes}\}$ as the set of all entities definable over a set of identifiers Id and
- $\mathcal{R}_{\mathcal{E}} := \{(e_1, e_2, relType) \mid e_1 = (eId_1, eType_1), e_2 = (eId_2, eType_2) \in \mathcal{E}, (eType_1, eType_2, relType) \in \text{RelationTypes}\}$ as the set of all relations definable over \mathcal{E}

Actors are users (or resources) who need privileges to work on certain tasks (e.g., workflow activities) or to access certain data objects (e.g., documents). Generally, access rules are not directly linked to actors, but to the more abstract concept of a *role*. Roles group privileges and are assigned to actors based on their capabilities and competences. Generally, an actor can play different roles: A physician in a hospital, for example, may play the two roles *ward doctor* and *radiologist*. Actors possessing the same role are considered as being interchangeable. Furthermore, roles can be hierarchically organized, i.e., a role may have

one or more specialized sub-roles. Thereby a sub-role inherits all privileges of its super-role and may extend this set by additional privileges. Finally, each actor can be assigned to an *organizational unit*. Like roles, organizational units can be hierarchically structured; i.e., a particular unit may have one or more subordinated units (e.g., a medical hospital may have an intensive care unit and an emergency laboratory).

Based on the introduced meta model we can now define the notion of *organizational model*. For the sake of simplicity, in this paper we do not consider the cardinalities associated with the relation types of our meta model (cf. Fig. 2).

Definition 1 (Organizational Model). *For the organizational meta model OMM let \mathcal{E} be the set of all entities over a given set of identifiers and let $\mathcal{R}_{\mathcal{E}}$ be the set of all relations over \mathcal{E} (see above). Then: An organizational model OM is defined as a tuple $(\text{Entities}, \text{Relations})$ with $\text{Entities} \subseteq \mathcal{E}$ and $\text{Relations} \subseteq \mathcal{R}_{\mathcal{E}}$.*

The set of all org. models definable on basis of OMM is denoted as \mathcal{OM} .

Let $OM = (\text{Entities}, \text{Relations})$ be an organizational model. Based on the organizational entities and relations described by OM we can define rules in order to control the access to tasks, services, documents, or other objects. Since the structuring and semantics of the access rules is fundamental for the (semi-) automated derivation of rule adaptations, we consider this issue in more detail. We distinguish between elementary and complex access rules.

An *elementary access rule* consists of a simple expression that qualifies a set of entities from OM (i.e., a subset of *Entities*) for this rule. The elementary access rule `Actor = 'Hunter'`, for example, expresses that exactly one entity, namely the actor with name 'Hunter', qualifies for this rule and therefore owns the privileges associated with it. As a second example consider the elementary access rule `OrgUnit = clinic`. For this access rule we denote the organizational unit `clinic` as the *qualifying entity*. Furthermore, all actors belonging to this unit own the privileges associated with this rule (e.g., getting access to a certain business document). For entities that can be hierarchically organized (i.e., for organizational units and roles in our meta model) we further allow for the definition of *transitive* elementary access rules. As an example consider the elementary access rule `OrgUnit = clinic(+)`. For this rule the set of qualifying entities comprises the organizational unit `clinic` itself and all of its directly or indirectly subordinated units (i.e., the *transitive closure* with respect to the 'is-subordinated' relation). All actors belonging to one of these qualifying units own the privileges associated with this rule. Similar considerations can be made regarding the 'specializes' relation between entities of type `Role`.

Definition 2 (Elementary Access Rule).

Let $OM = (\text{Entities}, \text{Relations})$ be an organizational model based on OMM. Then an elementary access rule EAR on OM is defined as follows:

$$\text{EAR} \equiv (\text{EAR1} \leftarrow (\text{EntityType} = e1)) \mid (\text{EAR2} \leftarrow (\text{OrgUnit} = e1(+))) \mid (\text{EAR3} \leftarrow (\text{Role} = e1(+)))$$

The set of entities qualifying for one of the elementary access rules EAR1, EAR2 or EAR3 can be determined as follows:

- $\text{EAR1} \leftarrow (\text{EntityType} = el)$
- $\text{QualEntities}(OM, \text{EAR1}) = \begin{cases} \{(el, \text{EntityType})\} & : (el, \text{EntityType}) \in \text{Entities} \\ \emptyset & : \text{otherwise} \end{cases}$
- $\text{EAR2} \leftarrow (\text{OrgUnit} = el(+))$
- $\text{QualEntities}(OM, \text{EAR2}) = \begin{cases} \{(el, \text{OrgUnit})\} \cup \text{Sub}(OM, el) & : (el, \text{OrgUnit}) \in \text{Entities} \\ \emptyset & : \text{otherwise} \end{cases}$
- with
- $\text{Sub}(OM, el) := \bigcup_{el':(el', el, \text{issubordinated}) \in \text{Relations}} (\{(el', \text{OrgUnit})\} \cup \text{Sub}(OM, el'))$
- $\text{EAR3} \leftarrow (\text{Role} = el(+))$
- $\text{QualEntities}(OM, \text{EAR3}) = \begin{cases} \{(el, \text{Role})\} \cup \text{Spec}(OM, el) & : (el, \text{Role}) \in \text{Entities} \\ \emptyset & : \text{otherwise} \end{cases}$
- with
- $\text{Spec}(OM, el) := \bigcup_{el':(el', el, \text{specializes}) \in \text{Relations}} (\{(el', \text{Role})\} \cup \text{Spec}(OM, el'))$

In order to enable the definition of more complex access rules we allow for the composition of existing rules (cf. Definition 3). For this purpose the following operators can be used: negation, conjunction, and disjunction.

Definition 3 (Access Rule).

Let $OM = (\text{Entities}, \text{Relations})$ be an organizational model based on OMM. Then an access rule AR on OM is defined as follows:

$$AR \equiv EAR \mid NEAR \mid CAR \mid DAR \text{ with}$$

- $NEAR \leftarrow (\text{NOT } (EAR))$ where EAR is an elementary access rule
- $CAR \leftarrow (AR1 \text{ AND } AR2)$ with AR1 and AR2 are access rules
- $DAR \leftarrow (AR1 \text{ OR } AR2)$ with AR1 and AR2 are access rules

Consider the organizational model OM depicted in Fig. 1. An example for a composed access rule on OM is $AR \leftarrow (\text{OrgUnit} = \text{medical clinic}(+) \text{ AND } \text{Role} = \text{assistant})$. Regarding the first part of this rule (i.e., the elementary access rule $EAR1 \leftarrow (\text{OrgUnit} = \text{medical clinic}(+))$) we obtain $\text{QualEntities}(OM, \text{EAR1}) = \{\text{medical clinic, treatment area}\}$ as the set of qualifying entities. For the second elementary rule $EAR2 \leftarrow (\text{Role} = \text{assistant})$ the set of qualifying entities is $\text{QualEntities}(OM, \text{EAR2}) = \{\text{assistant}\}$.

We can use Definition 2 and Definition 3 in order to determine the set of actors qualifying for access rule AR (cf. Definition 4). Corresponding actors then own the privileges associated with this rule.

Definition 4 (Valid Actor Set). Let $OM = (\text{Entities}, \text{Relations})$ be an organizational model. Let $\text{Act}(OM) := \{(a, \text{Actor}) | (a, \text{Actor}) \in \text{Entities}\}$ be the set of all actors defined by OM, and let AR be an access rule on OM. Then: $VAS(OM, AR)$ denotes the set of all actors (from OM) who qualify for AR (i.e., who own the privileges associated with rule AR). Formally:

$$- AR \leftarrow (\text{EntityType} = el) \implies$$

$$VAS(OM, AR) = \begin{cases} \{(el, \text{Actor}) | (el, \text{Actor}) \in \text{Act}(OM)\} & \text{if EntityType} = \text{Actor} \\ \{(a, \text{Actor}) | (a, \text{Actor}) \in \text{Act}(OM) \wedge \exists (a, el, \text{belongsto}) \in \text{Relations} | (a, el, \text{belongsto}) \in \text{Relations}\} & \text{if EntityType} = \text{OrgUnit} \\ \{(a, \text{Actor}) | (a, \text{Actor}) \in \text{Act}(OM) \wedge \exists (a, el, \text{has}) \in \text{Relations} | (a, el, \text{has}) \in \text{Relations}\} & \text{if EntityType} = \text{Role} \end{cases}$$

$$- AR \leftarrow (\text{EntityType} = el(+)) \implies$$

$$VAS(OM, AR) = \begin{cases} \{(a, Actor) | (a, Actor) \in Act(OM) \wedge \\ \exists el' \in QualEntities(OM, AR) : \\ \exists (a, el', belongsTo) \in Relations\} & \text{if } EntityType = OrgUnit \\ \{(a, Actor) | (a, Actor) \in Act(OM) \wedge \\ \exists el' \in QualEntities(OM, AR) : \\ \exists (a, el', has) \in Relations\} & \text{if } EntityType = Role \end{cases}$$

- $AR \leftarrow (\text{NOT } (AR1)) \text{ with } AR1 \text{ is EAR} \implies VAS(OM, AR) = Act(OM) \setminus VAS(OM, AR1)$
- $AR \leftarrow (AR1 \text{ AND } AR2) \text{ with } AR1 \text{ and } AR2 \text{ are access rules} \implies VAS(OM, AR) = VAS(AR1) \cap VAS(AR2)$
- $AR \leftarrow (AR1 \text{ OR } AR2) \text{ with } AR1 \text{ and } AR2 \text{ are access rules} \implies VAS(OM, AR) = VAS(AR1) \cup VAS(AR2)$

Finally, we provide a criterion which allows us to decide when an access rule AR is *valid* with respect to a given organizational model OM . We call an access rule valid if the following two conditions hold:

(1) AR does not contain *dangling references*, i.e., it does not refer to entities which are not present in OM . Formally:

$$DanglingRef(OM, AR) = \begin{cases} \text{False if } \forall \text{EAR in } AR : QualEntities(OM, EAR) \neq \emptyset \\ \text{True otherwise} \end{cases}$$

(2) AR is resolvable, i.e., the set of valid actors $VAS(OM, AR)$ does not become empty. Formally:

$$Resolv(OM, AR) = \begin{cases} \text{True if } VAS(OM, AR) \neq \emptyset \\ \text{False otherwise} \end{cases}$$

Note that dangling references and/or non-resolvable access rules might occur when changing organizational models in an uncontrolled manner.

Criterion 1 (Valid Access Rule) *Let $OM = (Entities, Relations)$ be an organizational model and let AR be an access rule on OM . Then AR is valid regarding OM if and only if there are no dangling references within the elementary access rules contained in AR and AR is resolvable over the set $Entities$, formally:*

$$Valid(OM, AR) = \text{True} \iff (DanglingRef(OM, AR) = \text{False} \wedge Resolv(OM, AR) = \text{True})$$

4 Organizational Evolution and Effects on Access Rules

How do changes of an organizational model OM affect the access rules based on it? In order to find a precise and satisfactory answer to this question, first of all, we must be able to formally define a model change Δ and its semantics (i.e., its effects on OM). Based on this information it should be possible to determine which access rules (on OM) are affected by the model change, how the effects of Δ on these rules look like, and which rule adaptations become necessary.

In the following we assume the scenario depicted in Fig. 3: A (correct) organizational model OM is transformed into another (correct) model OM' by applying change $\Delta = op_1, \dots, op_n$ to it. The challenge then is to adapt valid

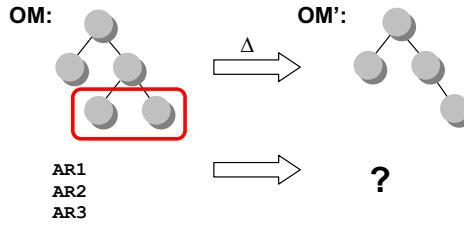


Fig. 3. Changing Organizational Models and Migrating Access Rules

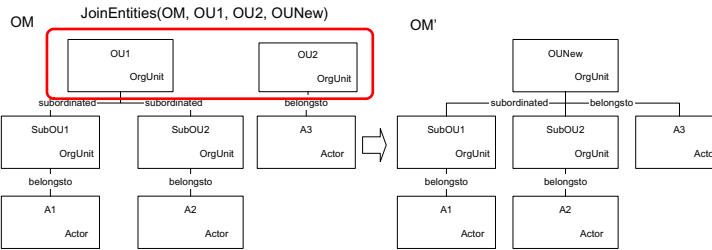


Fig. 4. Joining Organizational Units

access rules on OM in a way that they remain valid on OM' as well; i.e., to migrate these access rules from OM to OM' , but without causing errors or inconsistencies (e.g., dangling references).

In order to be able to express all kind of changes on an organizational model OM , we provide a complete set of basic change operations to the user (e.g., for creating / deleting entities and relations). For each change operation we define formal pre- and post-conditions, which preserve the correctness of OM when applying the operation(s) to it (assuming that OM has been a correct model before). In addition to these basic change operations we provide frequently used, high-level operations in order to facilitate change definition and to capture more semantics about changes. Examples include operations for joining two entities (e.g., organizational units; cf. Fig. 4) to a new one or for splitting an existing entity (e.g., a role) into two new entities. Formally:

Definition 5 (Change Framework for Organizational Models). Let \mathcal{E} be the set of all entities over a set of identifiers and let $\mathcal{R}_{\mathcal{E}}$ be the set of all relations over \mathcal{E} . Let further $OM = (Entities, Relations)$ be a (correct) organizational model. Table 1 defines basic change operations Δ , which transform OM into another (correct) organizational model $OM' := (Entities', Relations')$. In Table 2, in addition, two high-level change operations are given, whose definition is based on the basic change operations from Table 1.

Table 1. Basic Change Operations on Organizational Models

CreateEntity: $\mathcal{OM} \times Identifier \times EntityType \hookrightarrow \mathcal{OM}$ with $CreateEntity(\mathcal{OM}, eId, entityType) = \mathcal{OM}'$
Preconditions: • $(eId, entityType) \notin Entities$
Postconditions: • Entities' = $Entities \cup \{(eId, entityType)\}$ • Relations' = Relations
DeleteEntity: $\mathcal{OM} \times \mathcal{E} \hookrightarrow \mathcal{OM}$ with $DeleteEntity(\mathcal{OM}, e) = \mathcal{OM}'$
Preconditions: • $e \in Entities$ • $\exists rel = (e1, e2, relType) \in Relations$ with $e1 = e \vee e2 = e$
Postconditions: • Entities' = $Entities \setminus \{e\}$ • Relations' = Relations
CreateRelation: $\mathcal{OM} \times \mathcal{E} \times \mathcal{E} \times RelType \hookrightarrow \mathcal{OM}$ with $CreateRelation(\mathcal{OM}, e1, e2, relType) = \mathcal{OM}'$
Preconditions: • $e1 := (eId1, eType1), e2 := (eId2, eType2) \in Entities$ • $(e1, e2, relType) \in \mathcal{R}$ • $(e1, e2, relType) \notin Relations$
Postconditions: • Entities' = Entities • Relations' = $Relations \cup \{(e1, e2, relType)\}$
DeleteRelation: $\mathcal{OM} \times \mathcal{R}_{\mathcal{E}} \hookrightarrow \mathcal{OM}$ with $DeleteRelation(\mathcal{OM}, relation) = \mathcal{OM}'$
Preconditions: • relation $\in Relations$
Postconditions: • Entities' = Entities • Relations' = $Relations \setminus \{relation\}$
ReAssignRelation: $\mathcal{OM} \times \mathcal{R}_{\mathcal{E}} \times \mathcal{E} \times \mathcal{E} \hookrightarrow \mathcal{OM}$ with $ReAssignRelation(\mathcal{OM}, r, e, eNew) = \mathcal{OM}'$
Preconditions: • $r = (e1, e2, relType) \in Relations$ • $e = e1 \vee e = e2$ • $eNew := (eIdNew, eTypeNew) \in Entities$ • $e = e1 := (eId1, eType1) \implies eTypeNew = eType1$ • $e = e2 := (eId2, eType2) \implies eTypeNew = eType2$ • $e = e1 := (eId1, eType1) \implies (eNew, e2, relType) \notin Relations$ • $e = e2 := (eId2, eType2) \implies (e1, eNew, relType) \notin Relations$
Postconditions: • $e = e1 \implies Relations' = Relations \cup \{(eNew, e2, relType) \setminus \{(e1, e2, relType)\}$ • $e = e2 \implies Relations' = Relations \cup \{(e1, eNew, relType) \setminus \{(e1, e2, relType)\}$

A new relation (of type $relType$) between two entities $e1$ and $e2$ of an organizational model to $OM = (Entities, Relations)$, for example, can be created by applying the basic operation $CreateRelation(OM, e1, e2, relType)$ to OM . The pre-conditions associated with this operation then ensure that both entities $e1$ and $e2$ are actually present in OM and that the relation $(e1, e2, relType)$ is a valid relation not yet present in OM . The post-condition of this operation, in turn, describes the effects resulting from the application of the operation to OM (in our example, relation $(e1, e2, relType)$ is added to the set $Relations$, whereas the set $Entities$ remains unchanged).

When transforming an organizational model OM into another model OM' one must be able to decide which access rules defined on OM can be *directly migrated* to OM' , i.e., which rules can be immediately re-linked to the new model version without need for adaptation. Intuitively, this is the case for access rules which are also valid on OM' (cf. Theorem 1).

Theorem 1 (Direct Migration of Access Rules). *Let $OM = (Entities, Relations)$ be a (correct) organizational model. Let further AR be a valid access rule based on OM (i.e., $Valid(OM, AR) = True$) and let Δ be a (basic or high-level) change operation which transforms OM into another (correct) organizational model OM' . Then AR can be directly migrated to OM' if $Valid(OM', AR) = True$.*

Table 2. High-Level Change Operations on Organizational Models

JoinEntities: $\mathcal{OM} \times \mathcal{E} \times \mathcal{E} \times Identifiers \rightarrow \mathcal{OM}$ with $JoinEntities(\mathcal{OM}, e_1, e_2, nId) = \mathcal{OM}'$
Preconditions:
<ul style="list-style-type: none"> • $e_1 = (eId1, eType)$, $e_2 = (eId2, eType) \in Entities$ • $(nId, eType) \notin Entities$ • $eType \neq Actor$
Basic Change Operations:
<ul style="list-style-type: none"> • $CreateEntity(\mathcal{OM}, (nId, eType))$, $eNew := (nId, eType)$ • $\forall (e, e_1, relType) \in Relations: ReassignRelation(\mathcal{OM}, (e, e_1, relType), e_1, eNew)$ • $\forall (e, e_2, relType) \in Relations: ReassignRelation(\mathcal{OM}, (e, e_2, relType), e_2, eNew)$ • $\forall (e_1, e, relType) \in Relations: ReassignRelation(\mathcal{OM}, (e_1, e, relType), e_1, eNew)$ • $\forall (e, e_2, relType) \in Relations: ReassignRelation(\mathcal{OM}, (e, e_2, relType), e_2, eNew)$ • $DeleteEntity(\mathcal{OM}, e_1)$ • $DeleteEntity(\mathcal{OM}, e_2)$
SplitEntity: $\mathcal{OM} \times \mathcal{E} \times \mathcal{E} \times \mathcal{E} \rightarrow \mathcal{OM}$ with $SplitEntity(\mathcal{OM}, eOld, e_1, e_2) = \mathcal{OM}'$
Preconditions:
<ul style="list-style-type: none"> • $(eIdOld, eType) := eOld \in Entities$ • $(eId1, eType) := e_1, (eId2, eType) := e_2 \notin Entities$ • $eType \neq Actor$
Basic Change Operations:
<ul style="list-style-type: none"> • $CreateEntity(\mathcal{OM}, e_1)$ • $CreateEntity(\mathcal{OM}, e_2)$ • Reassignment of Relations — manually done by users • $DeleteEntity(\mathcal{OM}, eOld)$

The post conditions of the high-level changes result from the aggregation of the post conditions of the applied basic change operations.

Table 3. Preliminary Considerations – Adaptation of Access Rules

Let Δ be a change operation (cf. Def. 5) which transforms a (correct) org. model \mathcal{OM} into another (correct) org. model \mathcal{OM}' .

Δ

Necessary Adaptations

<code>CreateEntity(OM, e, eType)</code>	I, none
<code>CreateRelation(OM, e1, e2, relType)</code>	II, VAS may become bigger for EAR or smaller for NEAR ($= \emptyset$); no dangling references within AR
<code>DeleteRelation(OM, rel)</code>	II, VAS may become smaller for EAR ($= \emptyset$) or smaller for NEAR ; no dangling references within AR
<code>ReassignRelation(OM, rel, e, eNew)</code>	II, VAS may become smaller ($= \emptyset$) or bigger; no dangling references within AR
<code>DeleteEntity(OM, e)</code>	IV, VAS may become smaller ($= \emptyset$) for EAR or bigger for NEAR; dangling references within AR possible
<code>joinEntities(OM, e1, e2, eNew)</code>	IV, actor set changes, dangling references possible
<code>splitEntity(OM, e, e1, e2)</code>	IV, distribution of actor set is user dependent, dangling references possible

Regarding basic change operations the direct migration of access rules from \mathcal{OM} to \mathcal{OM}' is only possible in connection with the operation `CreateEntity(OM, ...)`, i.e., when adding new entities to \mathcal{OM} (cf. Lemma 1).

Lemma 1 (Direct Migration of Access Rules). *Let OM be a (correct) organizational model and let AR be a valid access rule on OM (i.e., $Valid(OM, AR) = \text{True}$). Let further Δ be a change operation which transforms OM into another (correct) organizational model OM' . Then AR can be directly migrated (re-linked) to OM' , i.e., $Valid(OM', AR) = \text{True}$ if $\Delta = \text{CreateEntity}(OM, \dots)$.*

When creating a new entity, we can always guarantee that there will be no dangling references within existing access rules and that this change is invariant regarding the set of valid actors (cf. Table 3). If an access rule AR cannot be directly transferred to the changed organizational model there may be two reasons for that. Either there are dangling references (e.g., due to the fact that an entity to which AR refers has been deleted from OM) or the set of valid actors becomes empty for AR on OM . The following lemma states for which basic change operations we can guarantee that there will be no dangling references within existing rules after a change (cf. Table 3).

Lemma 2 (No Dangling References). *Let OM be a (correct) organizational model and let AR be a valid access rule on OM (i.e., $Valid(OM, AR) = \text{True}$). Let further Δ be a change operation which transforms OM into another (correct) organizational model OM' . Then:*

DanglingRef(OM', AR) = False if

$\Delta \in \{\text{CreateEntity}(OM, \dots), \text{CreateRelation}(OM, \dots), \text{DeleteRelation}(OM, \dots), \text{ReAssignRelation}(OM, \dots)\}$.

For all other basic and high-level change operations, i.e., for changes $\Delta \in \{\text{DeleteEntity}(OM, \dots), \text{JoinEntities}(OM, \dots), \text{SplitEntity}(OM, \dots)\}$, their application to an organizational model OM may result in dangling references within the set of existing access rules (cf. Table 3).

In order to keep the total set of access rules consistent and the respective security component correctly running after applying model changes, we have to adapt those access rules that are no longer valid. Due to the potentially high number of rules that may exist in the system we want to assist the user as much as possible in accomplishing this task. In particular, we aim at the (semi-) automated migration and transformation of access rules in order to adapt them to model changes (if possible and meaningful). With 'semi-automated' we mean that the system shall assist the user in an adequate way, e.g., by exploiting the semantics of the applied change operation(s) and by making suggestions about potential rule transformations.

Theorem 2 indicates which rule adaptations can be automatically derived and suggested to the user in connection with the two high-level change operations presented before (cf. Table 2). In particular, Theorem 2 summarizes adaptation policies that can be applied to access rules containing dangling references. Doing so, our approach makes use of the semantics of the applied changes operations. Note that the derived policies only constitute suggestions, i.e., users may apply another strategy if more favorable.

Theorem 2 (Adaptation of Access Rules).

Let $OM = (\text{Entities}, \text{Relations})$ be a (correct) organizational model and let AR be a valid access rule on OM . Let further Δ be a change operation which transforms OM into another (correct) model OM' . Then: AR can be transformed into a valid access rule AR' on OM' by applying adaptation rule δ_{AR} (see below) if $\Delta \in \{\text{JoinEntities}(OM, \dots), \text{SplitEntity}(OM, \dots)\}$. For respective Δ adaptation rule δ_{AR} turns out as follows:

- $\Delta = \text{JoinEntities}(OM, e1, e2, \text{newE}) \implies \delta_{AR}:$
 - $\forall \text{EAR in } AR \text{ with EAR} := (\text{EntityType} = e1) \vee \text{EAR} := (\text{EntityType} = e2)$
replace EAR by $\text{EAR}' := (\text{EntityType} = \text{newE}) \wedge$
 - $\forall \text{EAR in } AR \text{ with EAR} := (\text{EntityType} = e1(+)) \vee \text{EAR} := (\text{EntityType} = e2(+))$
replace EAR by $\text{EAR}' := (\text{EntityType} = \text{newE}(+))$
- $\Delta = \text{SplitEntity}(OM, e, e1, e2) \implies \delta_{AR}:$
 - $\forall \text{EAR in } AR \text{ with EAR} := (\text{EntityType} = e)$
replace EAR by $\text{EAR}' := ((\text{EntityType} = e1) \text{ OR } (\text{EntityType} = e2)) \wedge$
 - $\forall \text{EAR in } AR \text{ with EAR} := (\text{EntityType} = e(+))$
replace EAR by $\text{EAR}' := ((\text{EntityType} = e1(+)) \text{ OR } (\text{EntityType} = e2(+)))$

As an example consider the change scenario depicted in Fig. 1. Take access rule $AR_{OM} := ((\text{OrgUnit} = \text{treatment area}) \text{ AND } (\text{Role} = \text{assistant}))$ and change Δ . For the first change operation $\text{joinEntities}(\text{PM}, (\text{treatment area}, \text{OrgUnit}), (\text{administration}, \text{OrgUnit}), (\text{patient services}, \text{OrgUnit}))$ the transformation described by Theorem 2 is applied and the access rule is transformed into $AR'_{OM'} = ((\text{OrgUnit} = \text{patient services}) \text{ AND } (\text{Role} = \text{assistant}))$. According to Theorem 2 deleting the two relations does not have any effect on the access rule. The deletion of entity $(\text{Hunter}, \text{Actor})$ is uncritical since the set of valid actors for AR' on OM' is non-empty.

Note that for join, split, and delete operations access rule transformations do not always become necessary. If an access rule does not refer to any entity joined, deleted, or splitted, the rule can stay unaltered after the respective model transformation. Finally, in addition to the described rule transformations in our current implementation we apply a number of other rule optimizations when migrating rules to a new version of the organizational model. The treatment of these optimizations, however, is outside the scope of this paper.

Our approach also deals with the challenging question of how to adapt access rules when entities are deleted from OM . As already mentioned this might lead to dangling references depending on the nature of the respective access rules. In certain cases no automatic strategy for adapting a particular access rule can be provided; the system then only reports the problem to the user and asks him for an adequate solution strategy. However, there are also many cases where automatic adaptations become possible, and thus users can be assisted in transforming rules in a way such that they become valid on the new model version OM' as well. In particular, this possibility exists in connection with the migration of composed access rules. As an example take access rule $(AR \leftarrow \text{Role} = R1 \vee \text{Role} = R2)$. If role $R2$ is deleted from the underlying organizational model this causes a dangling reference in AR . However, a suggestion for an automatic

adaptation would be to delete $\text{EAR} \leftarrow \text{Role} = \text{R2}$ from AR which results in the following rule: $\text{AR} \leftarrow \text{Role} = \text{R1}$. This access rule does note contain dangling references and it remains resolvable on OM' .

5 Practical Issues

To illustrate our results we apply them to an important use case of cooperative information systems – the adaptation of actor assignments in workflow management systems. More precisely we sketch how organizational changes are handled in the ADEPT2 process management system (PMS) [27] and how the different system components interact with each other to cope with model changes. Besides dynamic adaptations of organizational models ADEPT2 provides sophisticated support for process schema evolution [28] and process instance changes [7]. These kinds of process changes have been subject of previous publications on ADEPT and are outside the scope of this paper.

In the ADEPT2 buildtime environment, organizational models can be created and modified by the use of a graphical editor. This tool, whose visualization component is based on SVG (Scalable Vector Graphics), supports users in graphically defining organizational models. Model changes can be accomplished with the same tool and are based on the operations presented in this paper. All changes are logged by a respective system component. Besides this organization modeling component another tool exists, which allows users to define elementary and complex access rules based on the current version of the organizational model. Only such rules can be expressed which are syntactically and semantically correct. Furthermore, this rule editor is realized as plug-in which can be used within different client applications (in our case, for example, the workflow editor makes use of this component for defining actor assignments).

Consider the scenario depicted in Fig. 5. When an organizational change occurs authorized users can adapt the organizational model accordingly. In this case a new version of the organizational model is created which, in turn, triggers the migration and adaptation of related access rules. Rules which can be directly accessed by the change manager are immediately checked and migrated to the new model version. Rules which are kept outside the ADEPT2 system (e.g., access rules within a document management systems), however, require lazy migration techniques and more advanced mechanisms as well. In ADEPT2, any change of the organizational model immediately triggers the migration and adaptation of the access rules maintained within the ADEPT2 system. These rules include, for example, actor assignments for process activities (i.e., execution rights for working on these activities [24]) as well as privileges for changing process models and/or process instances [14].

In our scenario from Fig. 5, for example, a change of the depicted organizational model may require the adaptation of actor assignments within a process model. ADEPT2 indicates necessary adaptations to the process engineer who then can perform the respective changes at the process model level. Thereby we make use of the conceptual framework presented in this paper.

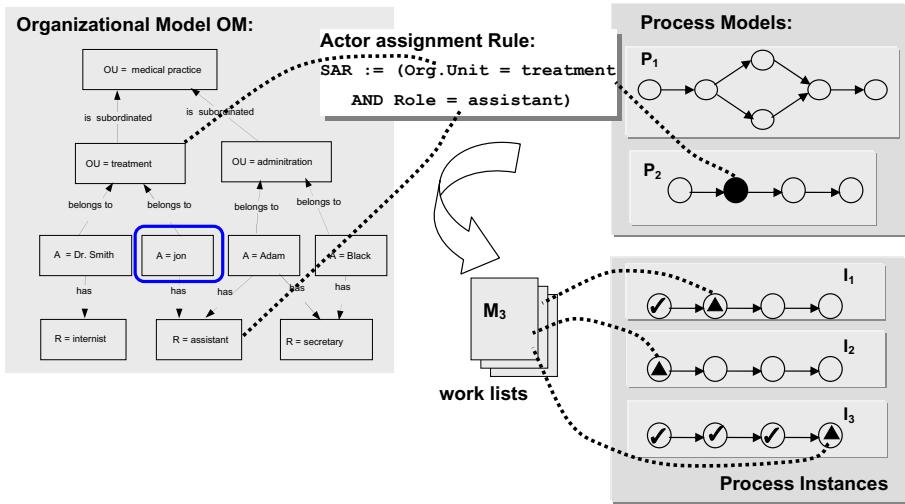


Fig. 5. Change Scenario in Process Management Systems

The needed rule adaptations (i.e., adaptations of actor assignments) are first carried out at the process model level. For long-running processes it might also become necessary to propagate these model changes (i.e., the adaptations of activity actor assignments) to already running process instances. This is only possible for process instances that are compliant with the current process model change. For example, when activities with modified actor assignment have not yet been activated such a change propagation is possible in ADEPT2. Finally, the described model and instance changes may also require the update of user worklists. This is one of the biggest challenges when thinking of realistic scenarios with ten thousands up to millions of work items. Worklist adaptations, however, are outside the scope of this paper. An overview of the scenarios supported in connection with changes of organizational models is given in Table 4.

Fig. 6 gives an overview of the different system components of ADEPT2 and also indicates the complexity arising from the design and implementation of adaptive process management technology. In the scenario described above the following components are involved: Editor, ChangeMgr, WorklistMgr, OrgModelMgr, AccessControlMgr, and LogMgr. Due to lack of space we omit a description of the architecture of this system and refer the interested reader to [27].

Table 4. Possible Scenarios when Changing the Organizational Model

Access Rule		Valid Actor Set
<i>not directly affected</i>	<i>unchanged</i>	<i>changed</i>
<i>directly affected</i>	<i>I</i> <i>III</i>	<i>II</i> (adapt worklists) <i>IV</i>

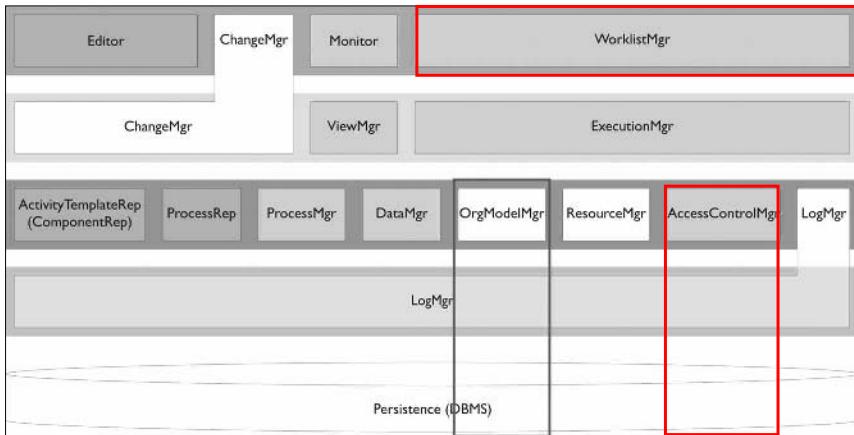


Fig. 6. ADEPT2 system architecture (abstract view)

6 Summary and Outlook

Both the controlled evolution of organizational models and the correct adaptation of access rules will be key ingredients of next generation enterprise security systems, ultimately resulting in highly adaptive access control models. Together with our complementary work on process evolution and dynamic process changes [7, 9, 29, 28] the presented concepts contribute to a platform enabling the realization of highly flexible and adaptive, cooperative information systems.

In this paper we have focussed on the common support of changes on organizational models and on the necessary adaptations of related access rules. We have discussed important challenges and requirements for the evolution of organizational models as well as the limitations of current approaches. The very important aspect of our work is its formal foundation. We have given precise definitions and formal theorems which are fundamental for the correct handling of model changes and adaptations of corresponding access rules. The treatment of both elementary and composed access rules adds to the overall completeness of our approach. Finally, in our ADEPT2 project a powerful proof-of-concept prototype has been implemented, which demonstrates the feasibility of the presented concepts. This prototype even uses a more expressive meta model to describe organizational structures in a compact and user-friendly way. Furthermore it considers the dynamic adaptation of actor assignments in process management systems and necessary worklist updates as well.

There are many other challenging issues related to changes of organizational models and of related access rules. First, we believe that respective changes must be closely linked with other components of cooperative information systems. For example, actor assignments in workflow-based applications may have to be adapted on-the-fly in order to cope with organizational changes. This, in turn, may require change propagation to hundreds up to thousands of in-progress

process instances as well as to related user worklists. Doing this in a correct and efficient manner is a non-trivial problem that will be investigated by us in more detail in future. Finally, changes may not only concern the process model or the organizational model but other components of the cooperative information systems as well. As an example take resource models or data models, which may be also subject of change.

References

1. v.d. Aalst, W., van Hee, K.: Workflow Management. MIT Press (2002)
2. Sutton, M.: Document Management for the Enterprise: Principles, Techniques and Applications. John Wiley (1996)
3. Linthicum, D.: Enterprise Application Integration. Addison-Wesley (1999)
4. Konyen, I.: Organizational structures and business processes in hospitals. Master's thesis, University of Ulm, Computer Science Faculty (1996) (in German).
5. Jablonski, S., Schlundt, M., Wedekind, H.: A generic component for the computer-based use of organizational models (in german). Informatik Forschung und Entwicklung **16** (2001) 23–34
6. Klarmann, J.: A comprehensive support for changes in organizational models of workflow management systems. In: Proc. 4th Int'l Conf. on Inf Systems Modeling (ISM'01). (2001) 375–387
7. Reichert, M., Dadam, P.: ADEPT_{flex} - supporting dynamic changes of workflows without losing control. JIIS **10** (1998) 93–129
8. Rinderle, S., Reichert, M., Dadam, P.: On dealing with structural conflicts between process type and instance changes. In Desel, J., Pernici, B., Weske, M., eds.: Proc. 2nd Int'l Conf. on Business Process Management (BPM'04). LNCS 3080, Potsdam, Germany (2004) 274–289
9. Rinderle, S., Reichert, M., Dadam, P.: Disjoint and overlapping process changes: Challenges, solutions, applications. In: Proc. Int'l Conf. on Cooperative Information Systems (CoopIS'04). LNCS 3290, Agia Napa, Cyprus (2004) 101–120
10. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. Data and Knowledge Engineering, Special Issue on Advances in Business Process Management **50** (2004) 9–34
11. Reichert, M., Rinderle, S., Dadam, P.: On the common support of workflow type and instance changes under correctness constraints. In: Proc. Int'l Conf. on Cooperative Information Systems (CoopIS'03). LNCS 2888, Catania, Italy (2003) 407–425
12. Bertino, E.: Data security. DKE **25** (1998) 199–216
13. zur Muehlen, M.: Resource modeling in workflow applications. In: Proc. of the 1999 Workflow Management Conference (Muenster). (1999) 137–153
14. Weber, B., Reichert, M., Wild, W., Rinderle, S.: Balancing flexibility and security in adaptive process management systems. In: Proc. Int'l Conf. on Cooperative Information Systems (CoopIS'05), Agia Napa, Cyprus (2005)
15. Ferraiolo, D., Kuhn, D., Chandramouli, R.: Role-Based Access Control. Artech House (2003)
16. NIST: Proposed Standard for Role-Based Access Control.
<http://csrc.nist.gov/rbac/rbacSTDACM.pdf> (2004)
17. Ferraiolo, D., Kuhn, D.: Role based access control. In: 15th National Computer Security Conference. (1992)

18. Sutton, M.: Document Management for the Enterprise – Principles, Techniques, and Applications. Wiley Computer Publ., New York (1996)
19. Botha, R., Eloff, J.: A framework for access control in workflow systems. *Information Management and Computer Security*. **9** (2001) 126–133
20. Bertino, E., Ferrari, E., Alturi, V.: The specification and enforcement of authorization constraints in wfms. *ACM Trans. on Inf. and Sys. Sec.* **2** (1999) 65–104
21. Wainer, J., Barthelmess, P., Kumar, A.: W-RBAC – a workflow security model incorporating controlled overriding of constraints. *International Journal of Collaborative Information Systems* **12** (2003) 455–485
22. Klarmann, J.: A comprehensive support for changes in organizational models of workflow management systems. In: Proc. Int'l Conf. on Information Systems Modeling (ISM'01), Hradec nad Moravici, Czech Republic (2001)
23. Domingos, D., Rito-Silva, A., Veiga, P.: Authorization and access control in adaptive workflows. In: Proc. Europ. Symposium on Research in Computer Science (ESORICS'03), Gjovik, Norway (2003) 23–28
24. Berroth, M.: Design of a component for organizational models. Master's thesis, University of Ulm, Computer Science Faculty (2005) (in German).
25. v.d. Aalst, W., Jablonski, S.: Dealing with workflow change: Identification of issues an solutions. *Int'l Journal of Comp. Systems, Science and Engineering* **15** (2000) 267–276
26. Klarmann, J.: Using conceptual graphs for organization modeling in workflow management systems. In: Proc. Conf. Professionelles Wissensmanagement (WM'01). (2001) 19–23
27. Reichert, M., Rinderle, S., Kreher, U., Dadam, P.: Adaptive process management with adept2. In: Proc. 21st Int'l Conf. on Data Engineering (ICDE'05), Tokyo (2005) 1113–1114
28. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. *Distributed and Parallel Databases* **16** (2004) 91–116
29. Rinderle, S., Weber, B., Reichert, M., Wild, W.: Integrating process learning and process evolution - a semantics based approach. In: 3rd Int'l Conf. on Business Process Management (BPM'05), Nancy, France (2005)

Towards a Tolerance-Based Technique for Cooperative Answering of Fuzzy Queries Against Regular Databases

Patrick Bosc, Allel Hadjali, and Olivier Pivert

IRISA/ENSSAT, 6 rue de kérampont - BP 80518
22305 Lannion Cedex, France
{bosc, hadjali, pivert}@enssat.fr

Abstract. In this paper, we present a cooperative approach for avoiding empty answers to fuzzy relational queries. We propose a relaxation mechanism generating more tolerant queries. This mechanism rests on a transformation that consists in applying a tolerance relation to fuzzy predicates contained in the query. A particular tolerance relation, which can be conveniently modeled in terms of a parameterized proximity relation, is discussed. The modified fuzzy predicate is obtained by a simple arithmetic operation on fuzzy numbers. We show that this proximity relation can be defined in a relative or in an absolute way. In each case, the main features of the resulting weakening mechanism are investigated. We also show that the limits of the transformation, that guarantee that the weakened query is not semantically too far from the original one, can be handled in a non-empirical rigorous way without requiring any additional information from the user. Lastly, to illustrate our proposal an example is considered.

1 Introduction

There has been an increasing interest for intelligent information systems endowed with cooperative behavior since the early' 90s. Such systems mainly intend to produce correct, non-misleading and useful answers, rather than literal answers to the users' queries [13]. These returned responses to queries allow to better serve the user's needs, expectations and interests. They contain the information that the user actually seeks. The most well-known problem approached in this field is the "*empty answer problem*", that is, the problem of providing the user with some alternative data when there is no data fitting his query. Several approaches have been proposed to deal with this issue. Some of them are based on a *relaxation* mechanism that expands the scope of the query [12][8][3]. This allows the database to return answers related to the original user's query which are more convenient than an empty answer. Other approaches propose knowledge discovery based solutions to this problem, see [16].

On the other hand, relying on *fuzzy queries* has the main advantage of diminishing the risk of empty answers. Indeed, fuzzy queries are based on preferences and retrieve elements that are more or less satisfactory rather than necessarily ideal. However, it still may happen that the database does not have any element that satisfies, even partially, the criterion formulated by the user. Then, an additional relaxation level

must be performed on the fuzzy query to avoid such empty answers. This can be accomplished by replacing fuzzy predicates involved in the query with weakened ones. The resulting query is then less restrictive.

In the fuzzy framework, query weakening consists in modifying the constraints contained in the query in order to obtain a less restrictive variant. Such a modification can be achieved by applying a basic transformation to each predicate of the query. Note that research on query weakening has not received much attention in the fuzzy literature. Very few works are concerned with this problem. The study done by Andreasen and Pivert [1] is considered as pioneering in this area. This approach is based on a transformation that uses a particular *linguistic modifier*. In [17], the authors consider queries addressed to data summaries and propose a method based on a *specified distance* to repair failing queries. Let us also mention the experimental platform [7], called PRETI, in information processing which includes a flexible querying module that is endowed with an empirical method to avoiding empty answers to user's request expressing a search for a house to let. At the end of the paper, we will give more details about these works.

Starting with the idea that allowing for a possible weakening of a query is connected with the intuitive idea of introducing some *tolerance* into it, one way to perform query weakening is to apply a *tolerance relation* to the fuzzy terms involved in the query. A particular tolerance relation which is of interest in the context of query weakening is considered. This relation can be conveniently modeled by a parameterized *proximity* relation. This notion of proximity, which originates from qualitative reasoning about fuzzy orders of magnitude [9][14], has been applied to define a fuzzy set of values that are *close* to some real-valued x . Let us recall that there are two points of view which can be considered to compare numbers and thus orders of magnitude x and y on the real line. We can evaluate to what extent the difference $x - y$ is *large*, *small* or *close* to 0; this is the absolute comparative approach. Or, we may use relative orders of magnitude, i.e., evaluate to what extent the ratio x/y is *close* to 1 or not. This leads to two possible ways for defining a proximity relation: in a *relative* or in an *absolute* way.

In this paper, we propose an alternative method to transform a fuzzy predicate P into an enlarged one, with the objective to keep it *semantically close* to the initial one, and where the notion of proximity plays a central role. This proximity is intended for defining a set of predicates that are close, semantically speaking, to a given predicate P . This is why this notion appears appropriate in the perspective of relaxing queries. Nevertheless, the way to address the problem using this notion can be viewed as an original philosophy in query weakening, since known approaches proceed differently and are based on other concepts.

The main features of the weakening mechanism resulting from the use of each kind of proximity as a basis for the predicate transformation, are investigated in depth. We show that in some cases our approach provides *semantic limits* to the iterative weakening process in order to ensure that the modified query is *as close as possible* to the original user's query. Moreover, the approach satisfies the properties required for any weakening process as well, as we will see later.

The paper is structured as follows. The next section recalls the problem of fuzzy query weakening on the one hand, and introduces the definition of the notion of proximity by means of fuzzy relations on the other hand. Section 3 shows how the

proposed proximity relation can be used for generating more fuzzy permissive predicates and for achieving query relaxation in the case of single-predicate query. Section 4 describes the possible weakening strategies for multiple-predicate queries and presents an example to illustrate our proposal. In section 5, we provide a summary about the available works in fuzzy query weakening. Last, we conclude and attempt to outline some future works.

2 Background

In this section, we first introduce the problem of query weakening in the fuzzy setting. Then, we present the modeling of the proximity relation of interest for expressing the notion of tolerance.

2.1 Fuzzy Query Relaxation Problem

Fuzzy (or *flexible*) queries [5] are requests in which user's preferences can be expressed. The user does not specify crisp conditions, but soft ones whose satisfaction may be regarded as a matter of a *degree*. Then, (s)he can distinguish between acceptable and non-acceptable answers in a more refined way than with a strictly Boolean filter. Towards this end, vague predicates are allowed in the requests; such predicates are represented by means of fuzzy sets¹ and model gradual properties. A typical example of a fuzzy query is: "retrieve the employees in a Department which are *young* and *well-paid*", where the fuzzy predicates "*young*" and "*well-paid*" are defined by the user (examples are given in figure 1). As a consequence, the result of a query is no longer a set of selected elements but a set of discriminated elements according to their global satisfaction.

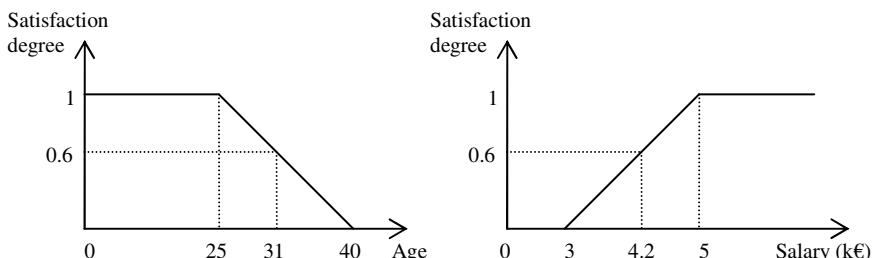


Fig. 1. Fuzzy predicates *young* and *well-paid* ($\mu_{\text{young}}(31) = \mu_{\text{well-paid}}(4.2) = 0.6$)

Introducing fuzziness in queries addressed to regular databases can be viewed as a primary level of relaxation and contributes to avoiding empty answers when classical

¹ A fuzzy set \mathcal{F} in the referential U is characterized by a membership function $\mu_{\mathcal{F}}: U \rightarrow [0, 1]$, where $\mu_{\mathcal{F}}(u)$ represents the grade of membership of u in \mathcal{F} . Two crisp sets are of particular interest when defining a fuzzy set \mathcal{F} : the core (i.e., $\text{Core}(\mathcal{F}) = \{u \in U / \mu_{\mathcal{F}}(u) = 1\}$) and the support (i.e., $S(\mathcal{F}) = \{u \in U / \mu_{\mathcal{F}}(u) > 0\}$).

crisp queries are too restrictive. Although, it still may happen that there is no elements in the database that satisfy, even partially, the vague criteria involved in a query. An additional weakening level is then necessary to return non-empty answers to the user's queries.

Weakening a "failing" fuzzy query consists in modifying the constraints involved in the query in order to obtain a less restrictive variant. Let Q be a fuzzy query of the form $P_1 \text{ and } P_2 \text{ and } \dots \text{ and } P_k$ (where P_i is a fuzzy predicate), and assume that the set of answers to Q is empty. A natural way to relax Q , in order to obtain a non-empty set of answers, is to apply a basic uniform transformation to each predicate P_i . This transformation process can be accomplished iteratively if necessary. Nevertheless, some desirable properties are required for any transformation T when applied to a predicate P :

(C_1): It does not decrease the membership degree for any element of the domain, i.e.

$$\forall u \in \text{domain}(A), \mu_{T(P)}(u) \geq \mu_P(u) \text{ where } A \text{ denotes the attribute concerned by } P;$$

(C_2): It extends the support of the fuzzy term P , i.e. $S(P) \subset S(T(P))$;

(C_3): It preserves the specificity of the predicate P , i.e. $\text{Core}(P) = \text{Core}(T(P))$.

Then, if P is a fuzzy predicate represented by the Trapezoidal Membership Function (TMF) (A, B, a, b) , the desired transformation T is such that

$$P' = T(P) = (A, B, T(A, a), T(B, b)), \quad (1)$$

where $[A, B]$ and $[A - T(A, a), B + T(B, b)]$ denote the core and the support of the modified predicate P' respectively. See figure 2.

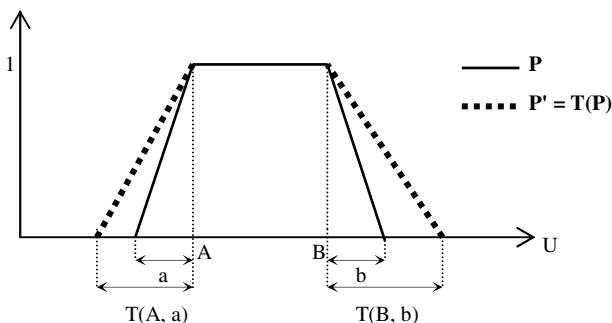


Fig. 2. Basic transformation

To be more efficient, a transformation T must still remain valid in the case of crisp predicates, i.e., predicates expressed in terms of traditional intervals. In other terms, the fuzzy inclusion $P \subset T(P)$ must still hold.

Besides, it is worthwhile that a transformation T allows for providing *semantic limits*. Namely, what is the maximum number of weakening steps that is acceptable according to the user, i.e., such that the final modified query is *not too far*, semantically speaking, from the original one. Such limits can offer a rational tool for controlling the relaxation process.

2.2 Modeling Tolerance

Let us first recall the formal definition of the concept of a tolerance relation [10]:

Definition. A tolerance relation (or proximity relation) is a fuzzy relation R on a domain U , such that for $u, v \in U$,

- (i) $\mu_R(u, u) = 1$ (reflexivity),
- (ii) $\mu_R(u, v) = \mu_R(v, u)$ (symmetry).

The properties of reflexivity and symmetry are very appropriate for expressing the degree of "*closeness*" or "*proximity*" between elements of a scalar domain. The quantity $\mu_R(u, v)$ evaluates the proximity, or similarity, between elements u and v . In the following we propose two ways for defining the notion of proximity on a scalar domain. We can evaluate to what extent $u - v$ is *close* to 0; this is the absolute comparative approach. Or, we may use relative orders of magnitude, i.e., we evaluate to what extent the ratio u/v is *close* to 1 or not.

2.2.1 Relative Closeness

A *relative closeness* relation (C_1) is a reflexive and symmetric fuzzy relation such that [14]:

$$\mu_{C_1}(x, y) = \mu_M(x/y), \quad (2)$$

where the characteristic function μ_M is that of a fuzzy number "*close to 1*", such that:

- i) $\mu_M(1) = 1$ (since x is close to x);
- ii) $\mu_M(t) = 0$ if $t \leq 0$ (assuming that two numbers which are close should have the same sign);
- iii) $\mu_M(t) = \mu_M(1/t)$ (since closeness is naturally symmetric). Then, M is a symmetric fuzzy number which means that $S(M) = [1 - \varepsilon, 1/(1 - \varepsilon)]$ with ε is a real number.

M is called a *tolerance parameter*. Strict equality is recovered for $M = I$ defined as $\mu_I(x/y) = 1$ if $x = y$ and $\mu_I(x/y) = 0$ otherwise. According to this point of view, we evaluate the extent to which the ratio x/y is close to 1. The closer x and y are, the closer to 1 x/y must be according to M . In the following $x \text{ } C_1[M] \text{ } y$ expresses that x and y satisfy the closeness relation in the sense of the fuzzy set M .

Semantic Properties of M . It has been demonstrated in [14] that the fuzzy number M which parameterizes closeness (and negligibility relation) should be chosen such that its support $S(M)$ lies in the validity interval $\mathbb{V} = [(\sqrt{5} - 1)/2, (\sqrt{5} + 1)/2]$ in order to ensure that the closeness relation be more restrictive than the relation "not negligible"². This means that if the support of a tolerance parameter associated with a closeness relation C_1 is not included in \mathbb{V} , then the relation C_1 is not in agreement with the intuitive semantics underlying this notion.

² The negligibility relation (Ne) is defined such that $\mu_{Ne[M]}(x, y) = \mu_{C_1[M]}(x+y, y)$. The following assumption holds as well: if x is close to y then neither is x negligible w.r.t. y , nor is y negligible w.r.t. x . The interval \mathbb{V} is such that $\mu_{C_1[M]}(x, y) \leq 1 - \max(\mu_{Ne[M]}(x, y), \mu_{Ne[M]}(y, x))$.

Interestingly enough, the validity interval \mathbb{V} will play a key role in the proposed query weakening process. As it will be shown later, it constitutes the basis for defining a stopping criterion of an iterative weakening process.

2.2.2 Absolute Proximity

An *absolute proximity* is an approximate equality relation which can be modeled by a fuzzy relation of the form [9]:

$$\mu_E(x, y) = \mu_Z(x - y), \quad (3)$$

which only depends on the value of the difference $x - y$, and where Z is a fuzzy set centered in 0, such that:

- i) $\mu_Z(r) = \mu_Z(-r);$
- ii) $\mu_Z(0) = 1;$
- iii) its support $S(Z) = \{r / \mu_Z(r) > 0\}$ is bounded and is denoted by $[-\delta, \delta]$ where δ is a real number.

Property (i) ensures the symmetry of the approximate equality relation ($\mu_E(x, y) = \mu_E(y, x)$); (ii) expresses that x is *approximately equal* to itself with a degree 1. Here we evaluate to what extent the amount $x - y$ is close to 0. The closer x is to y , the closer $x - y$ and 0 are. Classical equality is recovered for $Z = \mathbf{0}$ defined as $\mu_0(x - y) = 1$ if $x = y$ and $\mu_0(x - y) = 0$ otherwise. See [9] for other interesting properties of this relation. In the following $x E/Z y$ expresses that x and y satisfy E in the sense of Z .

3 Tolerance-Based Approach for Fuzzy Query Relaxation

3.1 Principle of the Approach

Starting with the idea that allowing for a possible weakening of a query is connected with the intuitive idea of introducing some tolerance into it, one way of performing a query weakening is to apply a tolerance relation to the fuzzy terms involved in the query. Let us consider a query which only involves one fuzzy predicate P , and a tolerance relation R . As stressed in section 2.1, relaxing Q consists in replacing the predicate P by an enlarged fuzzy predicate P' which can be defined as follows:

$$\forall u \in U, \mu_{P'}(u) = \sup_{v \in U} \min(\mu_P(v), \mu_R(u, v)). \quad (4)$$

Then, the basic transformation T is such that

$$P' = T(P) = P \circ R, \quad (5)$$

where \circ stands for the fuzzy composition operation³ [11]. Clearly, the tolerance-based transformation results in a modified predicate P' which gathers the elements of P and the elements outside P which are somewhat *close* to an element in P .

³ If R and S are two fuzzy relations on $U \times V$ and $V \times W$ respectively, $R \circ S$ is such that $\mu_{R \circ S}(u, w) = \sup_{v \in V} \min(\mu_R(u, v), \mu_S(v, w))$.

3.2 Non-symmetrical Relaxation

Let us first consider the case where the tolerance R is modeled in terms of the *relative closeness* relation, $Cl[M]$. Then, the following proposition holds (using (4)):

Proposition 1. Using the extension principle, the modified predicate P' is such that

$$P' = T(P) = P \otimes M,$$

where \otimes is the product operation extended to fuzzy numbers.

Proof. From (4), we have

$$\begin{aligned} \forall u \in U, \mu_{P'}(u) &= \sup_{v \in U} \min(\mu_P(v), \mu_{Cl[M]}(u, v)) \\ \mu_{P'}(u) &= \sup_{v \in U} \min(\mu_P(v), \mu_M(u/v)), \text{ since } \mu_{Cl[M]}(u, v) = \mu_M(u/v) \\ \mu_{P'}(u) &= \mu_{P \otimes M}(u), \text{ observing that } v \cdot (u/v) = u. \end{aligned}$$

Then, $P' = P \otimes M$. For more details about fuzzy arithmetic operations, see [11].

In terms of TMFs (Trapezoidal Membership Functions), if $P = (A, B, a, b)$ and $M = (I, I, \varepsilon, \varepsilon/(1 - \varepsilon))$ with $\varepsilon \in [0, (3 - \sqrt{5})/2]$, then $P' = (A, B, a + A \cdot \varepsilon, b + B \cdot \varepsilon/(1 - \varepsilon))$ by applying the above proposition. This leads to the following equalities $T(A, a) = a + A \cdot \varepsilon$ and $T(B, b) = b + B \cdot \varepsilon/(1 - \varepsilon)$ (see figure 1). It is easy to check that the desired properties (C_1) to (C_3) , required for any basic transformation, are satisfied in this case.

Now, by exploiting the above equalities, one can also easily perceive the meaning of the scalar $A \cdot \varepsilon$ (respectively $B \cdot \varepsilon/(1 - \varepsilon)$) which quantifies the left (respectively the right) weakening rate. Moreover, and since $\varepsilon < \varepsilon/(1 - \varepsilon)$ (for $0 < \varepsilon \leq (3 - \sqrt{5})/2 \approx 0.38$), the equality between $A \cdot \varepsilon$ and $B \cdot \varepsilon/(1 - \varepsilon)$ never holds, even when $A = B$, the resulting weakening is of a *non-symmetrical* nature.

In practice, if Q is a fuzzy query containing one predicate P (i.e., $Q = P$) and if the set of answers to Q is empty, then Q is relaxed by transforming it into $Q_1 = T(P) = P \otimes M$. This transformation can be repeated n times until the answer to the revised question $Q_n = T(T(\dots T(P)\dots)) = T^n(P) = P \otimes M^n$ is not empty.

Controlling Relaxation. In order to ensure that the revised query Q_n is semantically close enough to the original one, the support of M^n should be included in the interval

```

let Q = P
let ε be a tolerance value          (* ε ∈ [0, (3 - √5)/2] *)
i := 0                      (* i denotes the number of weakening steps *)
Qi := Q
compute ΣQi          (* ΣQi represents the set of answers to Qi *)
while (ΣQi = ∅ and S(Mi+1) ⊆ V) do
    begin
        i := i+1
        Qi := P ⊗ Mi
        compute ΣQi
    end
    if ΣQi ≠ ∅ then return ΣQi endif.

```

Algorithm 1.

of validity \mathbb{V} . In effect, the relation $\text{Cl}[M^n]$ is no longer a closeness relation semantically speaking when $S(M^n)$ does not lie in \mathbb{V} , despite its appearance at the syntactic level. Then, the above iterative procedure will stop either when the answer is non-empty or when $S(M^n) \not\subset \mathbb{V}$.

This weakening process can be formalized by algorithm 1 (the choice of ε will be discussed in section 4).

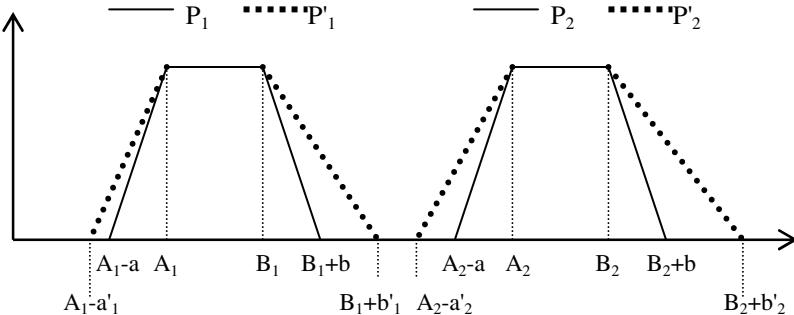


Fig. 3. Impact of the slopes and the relative position of the membership functions on the weakening mechanism when applying the closeness-based approach ($a = b$, $a'_1 = a + A_1\varepsilon$, $b'_1 = b + B_1\varepsilon/(1-\varepsilon)$, $a'_2 = a + A_2\varepsilon$, $b'_2 = b + B_2\varepsilon/(1-\varepsilon) \Rightarrow a'_1 < a'_2$, $b'_1 < b'_2$, $a'_i < b'_i$ for $i = 1, 2$)

Basic Properties. Let us now investigate the main features of this approach to single-predicate query weakening. As illustrated in figure 3, we can easily check that:

- i) The relative position of the membership function in the referential is a major factor that affects the weakening effect, rather than its left and right spreads. On the other hand, modifying the domain unit does not lead to any change in the weakening effect;
- ii) as already mentioned, this approach is basically *non-symmetric*. In figure 3, for instance, if $P_1 = (A_1, B_1, a, b) = (100, 180, 10, 10)$ then the weakened predicate $P'_1 = (A'_1, B'_1, a'_1, b'_1) = (100, 180, 10+10, 10+20)$ for $\varepsilon = 0.1$;
- iii) the Maximal Left Relaxation (MLR) is reached for the higher value of ε , namely $\varepsilon = \varepsilon_{max} = (3 - \sqrt{5})/2 (\cong 0.38)$. Then, MLR is equal to $\varepsilon_{max}A$ for any predicate $P = (A, B, a, b)$. As in the modified predicate $P' = (A, B, a', b')$, $b' = b + B\cdot\eta$ with $\eta = \varepsilon/(1 - \varepsilon)$, then the Maximal Right Relaxation (MRR) can be estimated by the following expression $B\cdot\eta_{max}$ with $\eta_{max} = \varepsilon_{max}/(1 - \varepsilon_{max}) = (\sqrt{5} - 1)/2 (\cong 0.61)$. This means that this approach provides *semantic limits* that may serve to control and restrict the weakening process.

3.3 Symmetrical Relaxation

This section is mainly concerned with the use of the *absolute proximity* relation, $E(Z)$, as a basis for modeling the tolerance relation R . Let us first state the following proposition which is straightforwardly obtained from (4):

Proposition 2. Using the extension principle, the modified predicate P' is such that

$$P' = T(P) = P \oplus Z,$$

where \oplus is the addition operation extended to fuzzy numbers.

Proof. Also from (4), we have

$$\begin{aligned} \forall u \in U, \mu_{P'}(u) &= \sup_{v \in U} \min(\mu_P(v), \mu_{E(Z)}(u, v)) \\ \mu_{P'}(u) &= \sup_{v \in U} \min(\mu_P(v), \mu_Z(u - v)), \text{ since } \mu_{E(Z)}(u, v) = \mu_Z(u - v) \\ \mu_{P'}(u) &= \mu_{P \oplus Z}(u), \text{ observing that } v + (u - v) = u. \end{aligned}$$

Then, $P' = P \oplus Z$.

As can be seen, the revised predicate P' contains P and the elements outside P which are in the neighborhood of an element of P . The basic transformation T writes $T(P) = P \circ E[Z] = P \oplus Z$. In terms of TMFs, if $P = (A, B, a, b)$ and $Z = (0, 0, \delta, \delta)$, then $P' = (A, B, a + \delta, b + \delta)$ using the above arithmetic formula. Clearly, this transformation is in agreement with the requirements (C_1) to (C_3) as well. Now according to figure 1, we have $T(A, a) = a + \delta$ and $T(B, b) = b + \delta$. This means that the weakening effect in the left and right sides is the same and is quantified by the scalar δ . Due to this equality, the resulting weakening is then of a *symmetrical* nature.

Now relaxing a query Q containing one predicate P ($Q = P$) can be achieved as follows. If the set of answers to Q is empty, then Q is transformed into $Q_1 = P \oplus Z$. This progressive relaxation mechanism can be applied iteratively until the answer to the revised query $Q_n = P \oplus nZ$ is not empty. From a practical point of view, this mechanism is very simple to implement, however, no information is provided about the *semantic limits*. Indeed, no intrinsic criterion is produced by this transformation enabling the stopping of the iterative process when the answer still remains empty.

Controlling Relaxation. To enable some controlling of the relaxation process, one solution consists in asking the user to specify, along with his query, the fuzzy set F_p of non-authorized values in the related domain. Then, the satisfaction degree of an element u becomes $\min(\mu_{Q_i}(u), 1 - \mu_{F_p}(u))$ with respect to the modified query Q_i

```

let Q := P
let δ be a tolerance value      (* Z = (0, 0, δ, δ) *)
i := 0
Qi := Q
compute ΣQi
while (ΣQi = ∅ and Core(S(Qi)) ⊈ Core(Fp)) do
begin
    i := i+1
    Qi := P ⊕ i·Z
    compute ΣQi
end
if ΣQi ≠ ∅ then return ΣQi endif.

```

Algorithm 2.

resulting from i weakening steps. The weakening process will now stop when the answer to Q_i is not empty ($\Sigma_{Q_i} \neq \emptyset$) or when the core of the complementary of the support of Q_i is included in the core of \mathbb{F}_P (i.e., $\min(\mu_{Q_i}(u), 1 - \mu_{\mathbb{F}_P}(u)) = 0$).

This weakening technique can be sketched by the algorithm 2 (where $\overline{S(Q_i)}$ denotes the complementary of the support of Q_i).

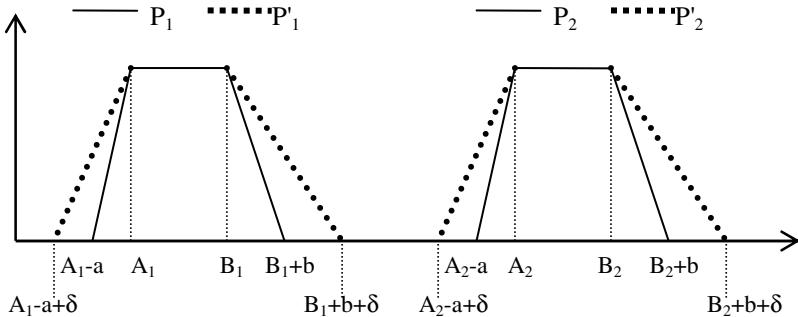


Fig. 4. Impact of the slopes and the relative position of the membership functions ($a_1=a_2=a < b_1=b_2=b \Rightarrow a'_1=a'_2=a+\delta < b_1=b_2=b+\delta$)

Basic Properties. Let us now take a look at the behavior of this query weakening method. From the above survey and as shown in figure 4, we point out that:

- i) the slopes and the relative position of the membership function have no impact on the weakening effect when using this approach; However, the attribute domain is identified as a major factor affecting the weakening because δ is an absolute value which is added and subtracted (δ will be different for the attributes "age" and "salary", for instance).
- ii) The nature of the resulting weakening is *symmetrical*. Indeed, the intensity of the weakening both in the left and right sides is given by the same quantity δ .
- iii) No means are provided for estimating in terms of the tolerance value δ , even roughly, the ratios *MLR* and *MRR* when using this weakening technique. However, we have $MLR = MRR$ in this case.

3.4 A Comparison

From a practical point of view, it is of great interest to compare the two proposed weakening methods in order to know in which case each may be the most suitable. To do this, we have listed five criteria that seem of major importance for the user:

- (i) *Preservation/modification of the specificity of the attribute.*
- (ii) *Symmetric/non-symmetric relaxation.*
- (iii) *Semantic control of the relaxation.*

- (iv) *Factors related to the domain and the predicate:* it consists in checking whether the attribute domain and the shape (or relative position) of the predicate membership function can have some impact on the weakening effect.
- (v) *Applicability in the crisp case:* is the transformation still valid for predicates expressed as traditional intervals?

In table 1, we summarize the behavior of each query weakening technique with respect to the above five criteria. As it is shown in the table, the most interesting features of the relative closeness-based approach is the rigorous semantic limits for controlling the query relaxation level that it provides. On the other hand, the benefit of the absolute proximity-based approach with respect to relative closeness-based one lies in the symmetrical nature of the weakening that it yields.

Table 1. A Comparison

Criteria	Relative closeness-based approach	Absolute proximity-based approach
(i)	Attribute specificity preserved	Attribute specificity preserved
(ii)	Non-symmetrical weakening	Symmetrical weakening by nature
(iii)	Semantic limits provided	No intrinsic semantic limits
(iv)	Attribute domain-independent and predicate membership function-dependent	Attribute domain-dependent and predicate membership function-independent
(v)	Still effective in the crisp case	Still effective in the crisp case

4 Relaxation of Complex Fuzzy Queries

A complex fuzzy query Q is of the form $P_1 \text{ op } P_2 \text{ op } \dots \text{ op } P_k$, where P_i is a fuzzy predicate and op stands for a connector which can express a *conjunction* (usually interpreted as a '*min*' in a fuzzy framework), a *disjunction* (usually interpreted as a '*max*'), etc.. An empty answer for a *disjunctive* query means that each predicate P_i has an empty support (relatively to the state of the database). The situation is similar when the connector, rather to be a *disjunction*, is a non-absorbing operator for zero (for instance the *mean*). On the other hand, for a *conjunctive* query, it suffices that one fuzzy term has an empty support so that the global answer is empty. In practice, this is the kind of queries that is problematic. This is why, and also for the sake of simplicity and brevity, we only consider conjunctive queries.

Let us first emphasize that the fuzzy set framework provides two types of approaches to weaken a complex fuzzy query: the *term modification-based* approach (that is the concern of this paper) and the *connector modification-based* approach. The latter is based on the replacement of one or more connectors by less restrictive

variants along the scale with disjunction as the least and conjunction as the most restrictive connector. For instance, if Q is a conjunctive query of the form $P_1 \text{ and } P_2 \text{ and } \dots \text{ and } P_k$ (where P_i is a fuzzy predicate), the idea is to replace one or several '*and*' operators by a less strict one. We will not consider this approach here. For more details, see [2].

4.1 Relaxation Strategies

In the case of the *term modification-based* approach, two options can be envisaged for the weakening procedure: i) a *global* query modification which consists in applying uniformly the basic transformation to all the terms in the query; ii) a *local* query modification which affects only some terms (or sub-queries). Most of the time, only a part of the query is responsible for the empty answers. As a consequence, it is not necessary to modify all the predicates in the query to avoid this problem. In such cases, local strategy seems more suitable and results in modified queries that are closer to the original one than the modified ones provided by the global strategy. Another argument in a favor of the local strategy is its ability for explaining the cause of the initial empty answer (indeed, only the modified predicates involved in the final revised query are responsible for the initial empty answer).

Global Strategy. As mentioned above, it consists in applying the basic transformation to the entire query. Given a transformation T and a conjunctive query $Q = P_1 \text{ and } P_2 \text{ and } \dots \text{ and } P_k$, the set of revised queries related to Q resulting from applying T is

$$\{T^i(P_1) \text{ and } T^i(P_2) \text{ and } \dots \text{ and } T^i(P_k)\},$$

where $i \geq 0$ and T^i means that the transformation T is applied i times. This strategy is simple but conflicts somewhat with our aim, that is, to find the closest revised queries.

Local Strategy. In this case, the basic transformation applies only to subqueries. Given a transformation T and a conjunctive query $Q = P_1 \text{ and } P_2 \text{ and } \dots \text{ and } P_k$, the set of modifications of Q by T is

$$\{T^{i_1}(P_1) \text{ and } T^{i_2}(P_2) \text{ and } \dots \text{ and } T^{i_k}(P_k)\},$$

where $i_h \geq 0$ and T^{i_h} means that the transformation T is applied i_h times. Assume that all conditions involved in Q are of the same importance for the user, a total ordering (\prec) between the revised queries related to Q can be defined on the basis of the number of the applications of the transformation T . Then, we have

$$Q' \prec Q'' \text{ if } \text{count}(T \text{ in } Q') < \text{count}(T \text{ in } Q'').$$

This ordering allows to introduce a *semantic distance* between queries.

For that semantic distance to make sense, it is desirable that T fulfills the property of *equal relaxation effect* on all terms. Several ways can be used for defining this property. A possible definition is to consider the ratio of the areas of the trapezes representing the membership functions associated to the original and the modified predicates. This ratio must be of the same magnitude when a certain transformation T is applied. Let us denote $\Delta(P, T(P))$ this ratio when T is applied to P . We have

$$\Delta(P, T(P)) = \mathcal{S}(T(P))/\mathcal{S}(P),$$

where $\mathcal{S}(P)$ and $\mathcal{S}(T(P))$ represent the areas of P and $T(P)$ respectively. A simple calculus enables to obtain (where $\eta = \varepsilon/(1 - \varepsilon)$ and $\Omega = B - A + (B + b) - (A - a)$):

$$\begin{aligned}\Delta(P, T(P)) &= 1 + (A \cdot \varepsilon + B \cdot \eta)/\Omega && \text{if } T(P) = P \circ \text{Cl}[M] \\ \Delta(P, T(P)) &= 1 + 2\delta/\Omega && \text{if } T(P) = P \circ E[Z].\end{aligned}$$

Now, given k predicates P_1, \dots, P_k , the equal weakening effect property for a set of transformation (T_1, \dots, T_k) can be expressed as follows:

$$\Delta(P_1, T_1(P_1)) = \Delta(P_2, T_2(P_2)) = \dots = \Delta(P_k, T_k(P_k)).$$

The total ordering induced by the transformation defines a lattice of modified queries. For instance, the lattice associated with the weakening of the query " $P_1 \wedge P_2 \wedge P_3$ " (with the symbol \wedge stands for the operator 'and') is given by figure 5:

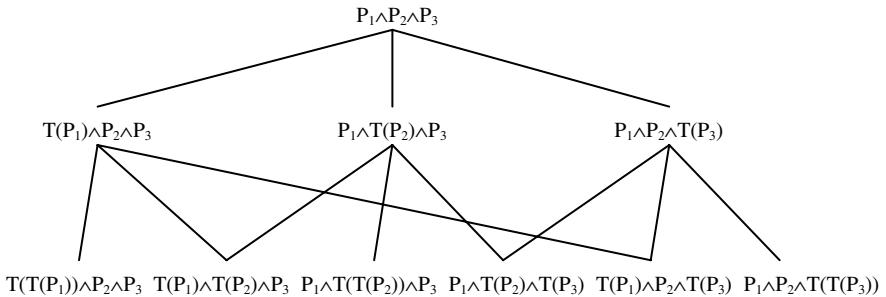


Fig. 5. Lattice of relaxed queries (reduced to two levels)

The above lattice can be explored as follows. We start by computing the satisfaction degrees of each predicate P_k ($k = 1, 3$). If at least one support is empty, we generate the weakened query by relaxing all the predicates with empty supports. Otherwise, we evaluate each weakened query belonging to the second level and we test the emptiness of its support. If all the answers are empty, we generate the weakened queries of the third level and the same method is applied and so on. To improve this strategy, one can use a measure of selectivity [15] as heuristic to guide through the lattice and to determine the terms which must be modified first. This notion gives an indication of the size of the answers to each weakened query without evaluating it against the database.

4.2 An Illustrative Example

A cooperative technique such as relaxation aims at finding a revised query semantically close to the user's query with a non-empty answer that has been obtained in the strict semantic limits of the relaxation process. As the relative closeness-based method allows for satisfying this goal by providing rigorous semantic limits that are a direct by-product of the fuzzy semantics underlying the closeness relation, we will illustrate our proposal using this method. Let us first make clear how some initial

conditions of this method could be set. Let us stress that some choices have been made only for the sake of illustration.

When using this query weakening method the first stage is to initialize the tolerant parameter ε . One way consists in asking the user to estimate the maximal number of weakening steps that he authorizes. Denoting by ω this number, the initial value of ε that we propose can be ε_{max}/ω (with $\varepsilon_{max} = (3 - \sqrt{5})/2 \approx 0.38$). We believe that this way of initializing ε could be acceptable by the user for two reasons. First, it takes his desires and requirements into account. Secondly, it ensures that the provided value lies in the authorized interval.

The proposed example concerns a user who wants to find the employees in a department who satisfy the condition: *young* and *well-paid*. The relation describing the considered employees is given in table 2.

Table 2. Relation of the employees

Name	Age	Salary (k€)	$\mu_{P_1}(u)$	$\mu_{P_2}(v)$
Dupont	46	3	0	0
Martin	44	2.5	0	0
Durant	28	1.5	0.8	0
Dubois	30	1.8	0.67	0
Lorant	35	2	0.34	0

Then, the query of interest writes $Q = "find \text{ employees } who \text{ are } young \text{ and well-paid"$ where *young* and *well-paid* are labels of fuzzy sets represented respectively by the TMFs $P_1 = (0, 25, 0, 15)$ and $P_2 = (5, +\infty, 2, +\infty)$ as drawn in figure 1. In the following, we will simply write $Q = P_1$ and P_2 .

As can be seen, each item of the database gets zero as satisfaction degree for the user's query Q with the content of table 2. This means that none of the items satisfies Q . Now, in order to return alternative answers to the user, we try to cooperate with him by relaxing his question. We first achieve this relaxation using the global strategy. Then, we consider the local strategy relaxation.

Table 3. First results based on global strategy

Name	Age	Salary (k€)	$\mu_{T(P_1)}(u)$	$\mu_{T(P_2)}(v)$	Satisfaction degrees to the relaxed query Q_1
Dupont	46	3	0	0.23	0
Martin	44	2.5	0	0.03	0
Durant	28	1.5	0.83	0	0
Dubois	30	1.8	0.72	0	0
Lorant	35	2	0.45	0	0

Global Strategy Relaxation. Assume that the user sets $\omega = 3$, then $\varepsilon = 0.12$ and the TMF of M is $(1, 1, 0.12, 0.13)$. By applying one weakening step, we transform Q into $Q_1 = T(P_1)$ and $T(P_2)$ where $T(P_1) = P_1 \otimes M = (0, 25, 0, 18.40)$ and $T(P_2) = P_2 \otimes M = (5, +\infty, 2.6, +\infty)$. Table 3 summarizes the returned results when querying the database using Q_1 . Unfortunately, the set of answers is still empty.

Then, a second weakening step is necessary to answer the initial user query. Since $S(M^2) = [0.77, 1.29]$ is included in \mathbb{V} , this step is acceptable from a semantic point of view. Now, the revised query is $Q_2 = T^2(P_1)$ and $T^2(P_2)$ where $T^2(P_1) = (0, 25, 0, 21.80)$ and $T^2(P_2) = (5, +\infty, 3.2, +\infty)$. This modification leads to the results reported in table 4.

As indicated in table 4, some employees of the database somewhat satisfy the weakened query Q_2 . Thus, the weakening process ends successfully and returns the set of answers to the user, i.e., the employees *Dupont*, *Martin* and *Lorant* with the satisfaction degrees 0.036 , 0.12 and 0.06 respectively.

Table 4. Final results based on global strategy

Name	Age	Salary (k€)	$\mu_{T^2(P_1)}(v)$	$\mu_{T^2(P_2)}(v)$	Satisfaction degrees to the relaxed query Q_2
Dupont	46	3	0.036	0.37	0.036
Martin	44	2.5	0.12	0.21	0.12
Durant	28	1.5	0.86	0	0
Dubois	30	1.8	0.77	0	0
Lorant	35	2	0.54	0.06	0.06

Local Strategy Relaxation. As shown in table 2, the support of the predicate P_2 is empty. Hence, P_2 is the responsible for the empty answer to the user's query Q . According to the way of building the lattice (see figure 5), the modification will firstly be applied to P_2 . The resulting query is then of the form $Q'_1 = P_1$ and $T(P_2)$ where $T(P_2) = (5, +\infty, 2.6, +\infty)$. As mentioned in the beginning of this section, this example only aims at illustrating our proposal. This why we have kept the same value for ε as above.

Table 5. First results based on local strategy

Name	Age	Salary (k€)	$\mu_{P_1}(u)$	$\mu_{T(P_2)}(v)$	Satisfaction degrees to the relaxed query Q'_1
Dupont	46	3	0	0.23	0
Martin	44	2.5	0	0.03	0
Durant	28	1.5	0.8	0	0
Dubois	30	1.8	0.67	0	0
Lorant	35	2	0.34	0	0

The summary of the returned answers to Q'_1 is given in table 5. As we can see, the set of answers still remains empty. An additional weakening step is then necessary to avoid such empty answers. By exploiting the lattice built on the basis of the revised queries related to Q , we have the choice between the modified queries $Q'_2 = P_1$ and $T^2(P_2)$ and $Q''_2 = T(P_1)$ and $T(P_2)$.

Let us consider for this weakening step the variant $Q'_2 = P_1$ and $T^2(P_2)$ where $T^2(P_2) = (5, +\infty, 3.2, +\infty)$. Table 6 gives the satisfaction degrees to Q'_2 of all the items contained in the database. As can be seen, the employee *Lorant* somewhat fulfils the vague requirements formulated in Q'_2 , then the relaxation process stops and *Lorant* is returned as an answer to the user.

Table 6. Final results based on local strategy

Name	Age	Salary (k€)	$\mu_{P_1}(u)$	$\mu_{T^2(P_2)}(v)$	Satisfaction degrees to the relaxed query Q'_2
Dupont	46	3	0	0.37	0
Martin	44	2.5	0	0.21	0
Durant	28	1.5	0.8	0	0
Dubois	30	1.8	0.67	0	0
Lorant	35	2	0.34	0.06	0.06

5 Related Work

To the best of our knowledge, not much work has been performed about fuzzy query relaxation. Andreassen and Pivert [1] have proposed an approach where the basic transformation is based on a particular *linguistic modifier*, called *v-rather*. To illustrate this approach, let us first consider a fuzzy predicate P represented by (A, B, a, b) . Enlarging P can be accomplished by applying a linguistic modifier to it. For example, the predicate "young" can be transformed into "more-or-less young" where *more-or-less* is an expansive modifier. Thus, in the case of the *v-rather* modifier, P can be replaced by the enlarged fuzzy predicate $P' = v\text{-rather}(P) = (A, B, a/v, b/v)$, with $v \in [1/2, 1]$.

Then, the basic transformation T is such that $T(A, a) = a/v$ and $T(B, b) = b/v$, see figure 1. We can also write $T(A, a) = a + \theta a$ and $T(B, b) = b + \theta b$, with $\theta = (1 - v)/v \in [0, 1]$. As can be seen, the resulting weakening effects in the left and right sides are obtained using the same parameter θ . This is why the approach is said to be *quasi-symmetric* (it is *symmetric*, if $a = b$). Furthermore, the requirements (C_1) - (C_3) , mentioned in section 2.1, are preserved by this modifier-based transformation.

In practice, assume that the user's query Q is such that $Q = P$. If there is no data that fit Q , it is transformed into $Q_1 = \text{rather}(P)$ and the process can be repeated n times until the answer to the question $Q_n = \text{rather}(\text{rather}(\dots \text{rather}(P)\dots))$ is not empty. The difficulty when applying this technique concerns its *semantic limits*. Indeed, no intrinsic information is provided about the criterion for stopping the

iterative process. To remedy this shortcoming, the authors advocate the use of the solution based on the fuzzy set \mathbb{F}_p of forbidden values (introduced in section 3.3). Besides, it is worthwhile to emphasize that this technique totally fails when relaxing classical crisp queries since $v\text{-rather}(P) = P$ if P is a crisp predicate.

In [17], the authors consider flexible queries addressed to data summaries and propose a method based on a specified distance to repair failing queries. If no summary fits a query Q , alternative queries are generated by modifying one or several fuzzy labels involved in Q . This requires a pre-established order over the considered attributes domains since a label is replaced by the closest one. The resulting queries are ordered according to their closeness to the original one taking into account the closeness of the modified labels. This distance-based measure of closeness is debatable since it does not consider the relative closeness between two labels. Moreover, this method does not provide any criterion for stopping the modification process if the answer is still empty.

Let us also mention the work done in the platform PRETI [7] which includes a flexible querying module. The user's request Q which expresses a search for a house to let, involves a set of preference profiles. Each profile P_i is modeled by means of fuzzy sets. If no answers are returned to the user, it is possible to avoid such empty answers by providing the closest answers to him. The idea is that each elementary requirement pertaining to an ordered domain is equipped with a default preference profile P'_i such that P'_i coincides with P_i on the values where $\mu_{P_i}(t) = 1$, and is non zero elsewhere on the whole attribute domain.

6 Conclusion

An alternative fuzzy set-based approach for handling query failure is proposed. It contributes to enrich cooperative answering techniques in the context of usual database fuzzy querying. The proposed method is based on the notion of proximity to define a basic predicate transformation. This transformation aims at finding a set of the closest predicates in the sense of the considered proximity. We have investigated the behavior of the weakening mechanism according to whether the transformation is based on a relative or an absolute proximity. We have shown that the former transformation allows providing rigorous semantic limits for controlling the relaxation process. This guarantees that the modified predicate is semantically close to the original one.

In case of a multi-predicate query, it would be desirable to extend that notion of *semantic closeness* to the overall query in order to asses the extent to which the final revised query (Q') is semantically close to the original query (Q). Of course, this measure of *semantic closeness* between Q' and Q could be established on the basis of the closeness measure of each original predicate and its weakened variant involved in Q' . One can evaluate the extent to which a predicate P is close to the modified predicate P' using the closeness measure of two imprecise values x and y (represented by two possibility distributions) related to the same attribute A , proposed in [6]. This constitutes one direction of our current work.

References

- [1] Andreasen T., Pivert O.: On the weakening of fuzzy relational queries. In Proc. of the 8th *Int. Symp. on Meth. For intell. Syst.*, Charlotte, USA, pp. 144-51, 1994.
- [2] Andreasen T., Pivert O.: Improving answers to failing fuzzy relational queries. In Proc. of the 6th *Int. Fuzzy Syst. Assoc. World Congress*, IFSA, São Paulo, Brazil, pp. 345-348, 1995.
- [3] Benamara, F., Saint Dizier, P.: Advanced relaxation for cooperative question answering. In *New Directions in Question Answering*, M. T. Maybury Ed., AAAI/MIT Press, 2004.
- [4] Bosc, P., HadjAli, A., Pivert, O.: Fuzzy closeness relation as a basis for weakening fuzzy relational queries. In Proc. of 6th *Inter. Conf. FQAS*, Lyon, France, 2004, pp. 41-53.
- [5] Bosc, P., Pivert O.: Some approaches for relational databases flexible querying. *Journal of Intelligent Information Systems*, 1, 1992, pp. 323-354.
- [6] Bosc, P., Pivert O.: On the comparison of imprecise values in fuzzy databases. In Proc. of 6th *IEEE Inter. Conf. on Fuzzy Systems*, Barcelona, Spain, July 1-5, 1997, pp. 707-712.
- [7] de Calmès, M., Dubois, D., Hullermeier, E., Prade, H., Sedes, F.: Flexibility and fuzzy case-based evaluation in querying: An illustration in an experimental setting. *Int. J. of Uncertainty, Fuzziness and Knowledge-based Systems*, 11(1), 2003, pp. 43-66.
- [8] Chu, W.W., Yang, H., Chiang, K., Minock, M., Chow, G., Larson, C.: CoBase: A scalable and extensible cooperative information system. *Journal of Intelligent Information Systems*, 6, No 1, 1996, pp. 223-259.
- [9] Dubois, D., HadjAli, A., Prade, H.: Fuzzy qualitative reasoning with words. In: *Computing with Words* (P.P. Wang, Ed.), John Wiley & Son, 2001, pp. 347-366.
- [10] Dubois D., Prade H.: Tolerant Fuzzy Pattern Matching: An Introduction. In: *Fuzziness in Database Management Systems* (Bosc P. and Kacprzyk J., Eds), Physica-Verlag, 1995.
- [11] Dubois D., Prade H.: *Fundamentals of Fuzzy Sets*. The Handbooks of Fuzzy Sets Series (Dubois D., Prade H., Eds), Vol. 3, Kluwer Academic Publishers, Netherlands, 2000.
- [12] Gaasterland, T.: Cooperative answering through controlled query relaxation. *IEEE Expert*, 12(5), Sep/Oct 1997, pp. 48-59.
- [13] Gaasterland, T., Godfrey, P., Minker, J.: An overview of cooperative answering. *Journal of Intelligent Information Systems*, 1(2), 1992, pp. 123-157.
- [14] HadjAli A., Dubois D., Prade H.: Qualitative reasoning based on fuzzy relative orders of magnitude. *IEEE Transactions on Fuzzy Systems*, Vol. 11, No 1, 2003, pp. 9-23.
- [15] Piatetsky-Shapiro, G., Connell, C.: Accurate estimation of the number of tuples satisfying a condition. In Proc. of ACM-SIGMOD, 1984, pp. 256-276.
- [16] Ras, Z.W., Dardzinska, D.: Failing queries in distributed autonomous information systems. In Proc. of ISMIS, LNAI No. 3488, 2005, pp. 152-160.
- [17] Voglozin, W.A., Rashia, G., Ughetto, L., Mouaddib, N.: Querying the SaintEtiq summaries: Dealing with null answers. In Proc. of 14th *IEEE Inter. Conf. on Fuzzy Systems*, Reno (Nevada), USA, May 22-25, 2005, pp. 585-590.

Filter Merging for Efficient Information Dissemination

Sasu Tarkoma and Jaakko Kangasharju

Helsinki Institute for Information Technology,
P.O. Box 9800, FIN-02015 HUT, Finland
`{sasu.tarkoma, jkangash}@hiit.fi`

Abstract. In this paper we present a generic formal framework for filter merging in content-based routers. The proposed mechanism is independent of the used filtering language and routing data structure. We assume that the routing structure computes the minimal cover set. It supports merging of filters from local clients, hierarchical routing, and peer-to-peer routing. The mechanism is also transparent and does not require modifications in other routers in the distributed system to achieve benefits. In addition to content-based routers, the system may also be used in firewalls and auditing gateways. We present and analyze experimental results for the system.

Keywords: Distributed event-based computing, publish/subscribe.

1 Introduction

Future applications and services are anticipated to require mechanisms for information processing, gathering, and distribution in dynamic environments. The popularity of information services that use *content delivery* and *content matching* using *interest profiles* or *filters* motivates the development of algorithms and protocols for efficient content dissemination, *publish/subscribe*, and profile processing. Example applications are news, stock market [1] and weather notification services, Internet games [2], group discussions and collaboration, and monitoring and controlling sensors and actuators.

Filtering is a central core functionality for realizing event-based systems and accurate content delivery. The main motivation for filtering is to improve accuracy in information delivery by delivering only those messages that are interesting for a client of the system — the delivered messages must match a priori filters defined by the client. Filters and their properties are useful for many different operations, such as matching, optimizing routing, load balancing, and access control. For example: *a firewall* is an example of a filtering router and *an auditing gateway* is a router that records traffic that matches the given set of filters. In content delivery, filters are continuous queries that are used to target content.

Filter merging is a technique to find the minimum number of filters and constraints that have maximal selectivity in representing a set of filters by modifying

constraints in the filters. Merging and covering are needed to reduce processing power and memory requirements both on client devices and on message or event routers. These techniques are typically general and may be applied to subscriptions, advertisements, and other information represented using filters. One of the main applications of filter merging is content-based routing in peer-to-peer systems.

In this paper, we present filter merging techniques for distributed *information routers*. A router is characterized using a routing data structure such as the Siena filters poset [3] or the poset-derived forest [4]. A content-based router accepts incoming messages and routes them to neighboring routers based on their interests. Interests are represented using filters or profiles defined in a multi-dimensional content space. A notification is typically a set of discrete values in this space and a filter is a set of discrete values or intervals.

The new contributions of this paper are as follows: 1. we present a formal framework for filter merging that is based on covering relations and is independent of the used filtering language and routing data structure, 2. the framework integrates with existing systems, such as the Siena event router, and 3. we present performance results on filter merging and discuss its feasibility.

The paper is structured as follows: In Section 2 we present the preliminaries and background. Section 3 presents the formal filter merging framework, and Section 4 examines the experimental results. Finally, in Section 5 we present the conclusions.

2 Background

The main functions of an information or event router are to match notifications for local clients and to route notifications to neighboring routers that have previously expressed interest in the notifications. The interest propagation mechanism is an important part of the distributed system and heart of the routing algorithm. The desirable properties for an interest propagation mechanism are small routing table sizes and forwarding overhead [5], support for frequent updates, and high performance.

2.1 Interest Propagation

In *subscription semantics*, the routers propagate subscriptions to other routers, and notifications are sent on the *reverse path* of subscriptions. This model may be optimized by constraining the propagation of subscriptions using advertisements. In *advertisement semantics* subscriptions are sent on the reverse path of overlapping advertisements. Many different interest propagation mechanisms have been proposed for both subscriptions and advertisements, including simple, covering, and merging based routing.

In *simple routing* each router knows all active subscriptions in the distributed system, which is realized by flooding subscriptions. In *identity-based routing* a subscription message is not forwarded if an identical message was previously forwarded. This requires an identity test for subscriptions. Identity-based routing removes duplicate entries from routing tables and reduces unnecessary forwarding of subscriptions.

In *covering-based routing* a covering test is used instead of an identity test. This results in the propagation of the most general filters that cover more specific filters. On the other hand, unsubscription becomes more complicated because previously covered subscriptions may become uncovered due to an unsubscription. *Merging-based routing* allows routers to merge exiting routing entries. Merging-based routing may be implemented in many ways and combined with covering-based routing [5]. Also, merging-based routing has more complex unsubscription processing when a part of a previously merged routing entry is removed.

2.2 Filter Model

We follow the basic concepts defined in the Siena system [3] and later refined and extended in Rebeca [6]. A filter F is a stateless Boolean function that takes a notification as an argument. Many event systems use the operators of Boolean logic, *AND*, *OR*, and *NOT*, to construct filters. A filtering language specifies how filters are constructed and defines the various predicates that may be used. A predicate is a language specific constraint on the input notification. Typically, filtering languages are compositional in the sense that, for example, a filter is composed from subfilters, which are defined using predicates. Predicates are called atomic when they are the smallest possible unit of composition.

A filter is said to match a notification n if and only if $F(n) = \text{true}$. The set of all notifications matched by a filter F is denoted by $N(F)$. A filter F_1 is said to cover a filter F_2 , denoted by $F_1 \sqsupseteq F_2$, if and only if all notifications that are matched by F_2 are also matched by F_1 , i.e., $N(F_1) \supseteq N(F_2)$. The \sqsupseteq relation is reflexive and transitive and defines a partial order. The filter F_1 is equivalent to F_2 , written $F_1 \equiv F_2$, if $F_1 \sqsupseteq F_2$ and $F_2 \sqsupseteq F_1$. A set of n filters $S_F = \{F_1, \dots, F_n\}$ covers a filter F if and only if $N(S_F) \supseteq N(F)$, i.e., $\bigcup_i^n N(F_i) \supseteq N(F)$. A set of filters is covered if each filter in the set is covered.

The covering test may be performed efficiently for simple filters; however, it becomes complicated for more complex filters. There are several solutions for processing complex filters, for example, they may be partitioned into several simple filters. Covering relations can also be computed offline and they may be approximated.

2.3 Data Structures

One of the first content-based routing data structures was presented in the Siena project [3]. The *filters poset* (partially ordered set) structure was used by event routers to manage subscriptions and advertisements from other routers. The structure stores filters by their covering relation and computes the most general set of filters that should be sent to neighboring routers. In Siena terminology, each filter has a set of subscribers and a set that contains the neighbors to which the filter has been sent (the *forwards* set). Filters poset computes the immediate predecessor and successor sets for each filter by walking through the structure. The predecessor set is used to determine if a covering filter has been forwarded

to a neighboring router. In this case, there is no need to forward the current filter. An unsubscription forwards the filter that needs to be removed using the *forwards* set and may result in a number of uncovered filters that must also be forwarded as subscriptions with the unsubscription.

In event literature filters that represent subscriptions and advertisements are typically manipulated as sets. The Siena filters poset was found to be limited in terms of scalability, which led to the development of the combined broadcast and content-based (CBCB) routing scheme [7]. We have previously proposed the poset-derived forest that addresses the scalability limitations of the filters poset by only storing subset of the covering relations [4]. We are not aware of other efficient data structures for processing frequent filter set additions and removals. The filter-based routing structures differ from database indexing algorithms, because they are organized using the covering relation and do not assume metric spaces. Indexing techniques, such as R-trees and interval trees [8], may also be used, but they introduce new assumptions on the nature of the filters.

We have developed a graphical tool, called the *PosetBrowser*¹, for experimenting with various content-based routing data structures. The PosetBrowser demonstrates four different data structures: the filters poset, the colored forest, the balanced colored forest, and the non-redundant balanced colored forest. A balanced forest uses the subscriber interfaces, called colors, to optimize insertions and matching. Redundancy means that there may be a filter associated with an interface that is covered by some other filter set associated with the same interface. In order to prevent false positives, non-redundancy is needed for hierarchical and peer-to-peer routing. The PosetBrowser also demonstrates the filter merging mechanism presented in Appendix A.

2.4 Filter Merging

A filter merging-based routing mechanism was presented in the Rebeca distributed event system [6]. The mechanism merges conjunctive filters using perfect merging rules that are predicate-specific. Routing with merging was evaluated mainly using the routing table size and forwarding overhead as the key metrics in a distributed environment. Merging was used only for simple predicates in the context of a stock application [5, 6]. The integration of the merging mechanism with a routing data structure was not elaborated and we are not aware of any results on this topic.

A mergeability rule was sketched in [6] by observing that a merged set of filters M created from a set of filters can be forwarded to all neighbors if any notification n that is matched by M is matched by at least one filter of a local client and by two distinct filters F_i and F_j with differing output interfaces. The *forwards* set-based formulation presented in this paper allows a more flexible and elegant way to determine mergeability.

The optimal merging of filters and queries with constraints has been shown to be NP-complete [9]. Subscription partitioning and routing in content-based

¹ Available at www.hiit.fi/fuego/fc/demos

systems have been investigated in [10, 11] using Bloom filters [12] and R-trees for efficiently summarizing subscriptions.

Bloom filters are an efficient mechanism for probabilistic representation of sets. They support membership queries, but lack the precision of more complex methods of representing subscriptions. To take an example, Bloom filters and additional predicate indices were used in a mechanism to summarize subscriptions [13, 14]. An Arithmetic Attribute Constraint Summary (AACS) and a String Attribute Constraint Summary (SACS) structures were used to summarize constraints, because Bloom filters cannot capture the meaning of operators other than equality. The subscription summarization is similar to filter merging, but it is not transparent. The event routers need to be aware of the summarization mechanism. In addition, the set of attributes needs to be known a priori by all brokers and new operators require new summarization indices. The benefit of the summarization mechanism is improved efficiency since a custom matching algorithm is used that is based on Bloom filters and the additional indices.

3 Merging Mechanisms

In this section we present techniques for incorporating filter merging into content-based routers in a transparent fashion. The techniques are independent of the used filtering language and routing data structure, and do not depend on the mechanism that is used to merge two input filters. We present two distinct ways to merge filters: local merging and remote merging. In the former, merged filters are placed into the data structure. In the latter, merged filters are stored separately from the data structure and only for exiting (outgoing) routing table entries.

3.1 Merging and Routing Tables

We propose a merging extension to the generic content-based routing table. The desired characteristics for this merging mechanism are simplicity in implementation, efficiency, and minimal requirements for the underlying routing table. We assume that a $\text{merge}(F_1, F_2)$ procedure exists that merges input filters F_1 and F_2 and returns a single merged filter F_M for which $F_M \sqsupseteq F_1$ and $F_M \sqsupseteq F_2$. A merge of two or more filters is called a *merger*. Filter merging is useful, because it allows to further remove redundancy and keep the number of elements minimal.

A merger is either *perfect* or *imperfect*. A perfect merger does not result in false positives or negatives, whereas an imperfect merger may result in false positives. In addition to accuracy, we have additional requirements with filter merging:

- Merging must be transparent for applications and routers.
- Merging must maintain the set of inserted nodes. An insert of x may result in a new merged node $\text{merge}(x,y)$, but after the delete of x the resulting node must cover y .

Filter merging may be applied in different places in the event router. We distinguish between three different merging scenarios and techniques: *local merging*, *root merging*, and *aggregate merging*. In the first scenario, filter merging is performed within a data structure. In the second scenario, filter merging is performed on the root sets of local filters, edge/border routers, and hierarchical routers. In the third scenario, filter merging is performed on the two first levels of a peer-to-peer data structure, such as the filters poset.

Figure 1 presents two router configurations with filter merging and highlights the modular structure of content-based routers. Subfigure I illustrates filter merging in peer-to-peer routing. The filters poset is an example data structure for peer-to-peer routing. Local clients are stored by the redundant colored forest data structure, which is the preferred structure for storing filters from local clients [4].

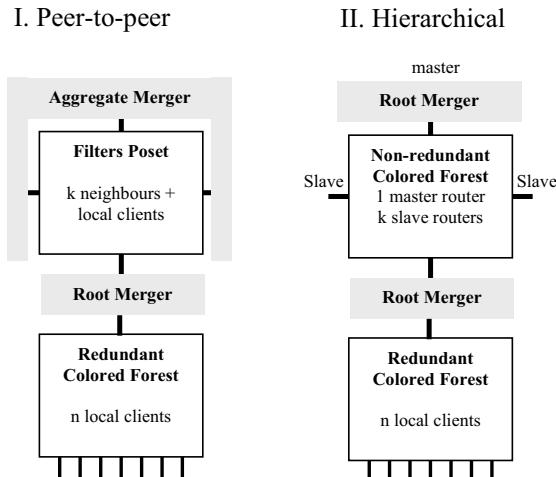


Fig. 1. Merging extension for routing tables

Two different merging techniques are used in the figure: root merging for local clients and aggregate merging for remote operation. The merging of the local filters is easy and efficient, because it is performed only on the root-set and re-merging is needed only when this set changes.

Subfigure II shows the use of filter merging in the hierarchical environment using the forest data structure. The figure illustrates the use of root merging for both local clients and the master router. Filter merging is easy for both local clients and the master router. Only the root sets of the local routing table and external routing table, a non-redundant colored forest, are merged. The forest is superior to the filters poset in hierarchical operation, because the computation of the *forwards* sets is not needed.

3.2 Rules for Merging

Two rule sets are needed in order to ensure that data structures that have been extended with merging are equivalent to the same data structures without merging. First, a set of *mergeability rules* are defined that specify when two filters may be merged. Then, we define a set of *merging rules* for preserving equivalence for insertions and deletions between routing data structures and their counterparts that have been extended with filter merging.

Mergeability Rules. Filters may be merged in two ways: *local merging* that is performed within the data structure and *aggregate merging* that is performed for exiting (outgoing) routing table entries only. The former is given by the local merging rule presented in Definition 1. This rule says that only those filters that are mergeable and share an element in the *subscribers* set may be merged. This requires that the *subscribers* sets of the input filters are updated accordingly. Local merging means that mergeable filters from the same interface are merged. This allows merged filters to be stored within the data structure. This approach puts more complexity into the data structure, but benefits also the local router.

Definition 1. *Local merging rule:* *The operation $\text{merge}(F_1, F_2)$ may be performed if F_1 and F_2 are mergeable and the intersection of their subscribers sets has at least one element, $\text{subscribers}(F_1) \cap \text{subscribers}(F_2) \neq \emptyset$. The subscribers set of the resulting merger must contain only a single element.*

The latter option is given by Definition 2. Aggregate merging merges any filters that have the same or overlapping *forwards* sets. This may be applied only to exiting entries and the mergers should not be placed into the data structure. Aggregate merging allows the aggregation of multicast traffic, since the *forwards* sets of root nodes are essentially sent to most neighbors. Only the first two levels of nodes need to be considered and in most cases it is enough to inspect only root nodes. On the other hand, this approach does not benefit the local router in matching operations, but the benefit is gained in distributed operation if all neighbors employ this approach as well.

Definition 2. *Aggregate merging rule:* *Given that the forwards sets of the filters are non-empty, the filters are mergeable only when $\text{forwards}(F_1) \cap \text{forwards}(F_2) \neq \emptyset$. The forwards set of the resulting merger is the intersection of the two forwards sets.*

The rule of Definition 1 corresponds to interface specific merging of filters. The rule of Definition 2 takes into account the *forwards* sets and aggregates multicast traffic. These rules may be applied simultaneously.

Local Merging Rules. Let MR denote the set of merged nodes/filters. Each element $x \in MR$ is a result of a sequence of *merge* operations and has a corresponding set, denoted by $CO(x)$, which contains the components of the

merger x . Further, let $CV(x)$ denote nodes that were removed due to covering by x if the merger is placed in the data structure. The sets CO and CV are needed in order to maintain transparent operation.

We present six rules for maintaining equivalence. These rules do not specify the semantics or performance of the merging mechanism. They specify the requirements for equivalence. A merging algorithm needs to follow these rules. For example, rule number five does not imply that re-merging of the removed merger should not be done. We assume that the routing data structure provides two operations: *add* and *del*. The *add* inserts a filter to the structure, and *del* removes a filter. Note that the *del* in rule four is applied to a merger, and the *del* in rule five is applied to a component of a merger. The *del* operation for a merger is only invoked internally; the client of the system that sent the components of the merger has no knowledge of its existence. When a *del* is performed to a node that is part of a merger's CV set, the deleted node is removed also from that set.

We also define two auxiliary operations *addComponent* and *addComponents*. The *addComponent*(S, F) operation takes a set S and a filter F as arguments and adds F to S if there does not exist a filter in S that covers F . Similarly, any filters in S covered by F are removed from S . The *addComponents*(S, P) operation is similar to *addComponent*, but the second argument, P , is a set.

The following rules assume that $\text{subscribers}(F_1) \supseteq \text{subscribers}(F_2)$ and that identical filters, $F_1 \equiv F_2$, are detected and processed before any merging rules are applied. The rules pertain to two arbitrary input filters F_1 and F_2 . The rules are presented as tautologies, they have to be always true, and we assume that each operation on the right side returns true. We assume that elements in a conjunction are evaluated from left to right.

1. $F_1 \sqsupseteq F_2 \wedge F_1 \notin MR \wedge F_2 \notin MR \Rightarrow \text{del}(F_2)$. This rule says that when a non-merged node covers another non-merged node, the covered node is removed.
2. $F_1 \sqsupseteq F_2 \wedge F_1 \notin MR \wedge F_2 \in MR \Rightarrow \text{del}(F_2)$. This rule states that when a merger is covered by a non-merger, the merger is removed and all of its components are also removed (Rule 6).
3. $F_1 \sqsupseteq F_2 \wedge F_1 \in MR \wedge F_2 \notin MR \Rightarrow \text{del}(F_2) \wedge \text{addComponent}(CV(F_1), F_2)$. This rule states that when a merger covers a non-merger, the covered node is removed and added to the merger's set of covered nodes.
4. $F_1 \sqsupseteq F_2 \wedge F_1 \in MR \wedge F_2 \in MR \Rightarrow \text{addComponents}(CO(F_1), CO(F_2)) \wedge \text{addComponents}(CV(F_1), CV(F_2)) \wedge \text{del}(F_2)$. Specifies that when a merger covers another merger, the covered merger is removed (Rule 6) and all components of the merger and nodes covered by the merger are added to the respective sets of the covering merger.
5. $\text{del}(F_1) \wedge (\exists x \in MR : F_1 \in CO(x))(\forall x \in CO(F_1) \setminus \{F_1\} : \text{add}(x)) \wedge (\forall x \in CV(F_1) : \text{add}(x))$. This rule says that when a component of a merger is removed, all the components and covered nodes should be returned to the data structure. After this, the merger should be removed.

6. $\text{del}(F_1) \wedge F_1 \in MR \Rightarrow (MR' = MR \setminus \{F_1\}) \wedge (\forall x \in CO(F_1) : \text{del}(x)) \wedge (\forall x \in CV(F_1) : \text{del}(x))$. This rule states that when a merger is removed, all its components must also be removed.

Aggregate Merging Rules. Aggregate filter merging rules are similar to the local merging rules with the exception that they do not need to address covered nodes, because merged filters are not inserted into the data structure. On the other hand, it is useful to keep track of the nodes covered by a merger, the direct successor set. This may be done by placing covered nodes in CO or by using the CV set. In the former case, the third rule is not needed, and for the latter case the del in the third rule is not performed. For aggregate merging, instead of the *subscribers* set condition, we have the *forwards* set condition presented in Definition 2. Implementations need to update the *forwards* sets of any mergers covered by other mergers.

3.3 A Generic Aggregate Mechanism

We present a simple generic remote merging mechanism based on the remote merging rules. The data structure must provide two information sets: the root set, and the *forwards* sets of root nodes. Both are easy to compute and the computation of the *forwards* set may be performed based on the root set alone. We propose that all mergeable root filters with the same *forwards* sets are merged. The root set is the natural candidate set for merging, because it covers other filters. By merging filters with the same *forwards* sets we simplify the finding of the mergeable set. Also, there is no need to keep track of separate *forwards* set entries.

The proposed technique may be applied to both hierarchical routing and peer-to-peer routing. For hierarchical routing, the *forwards* set of root nodes contains only a single entry, the master router. In peer-to-peer routing, merged sets are always multicast to at least $|neighbors| - 1$ external interfaces. This merging technique may be called *weakly merging* for peer-to-peer routing, because it does not merge all mergeable candidates and unicast updates are not considered. It is more efficient to operate on aggregates than on separate entries.

The proposed aggregate merging mechanism is:

Generic. It makes minimal assumptions on the underlying data structure. It may be used with both peer-to-peer and hierarchical routing, and also for local clients. Merging a filter in the hierarchical scenario is equivalent to merging filters from local clients, because in both cases there is only one direction where the messages are forwarded.

Efficient. It is activated only when the root set changes, and it uses the *forwards* sets to aggregate merger updates. This kind of approach may be used to leverage any multicast mechanisms.

Relatively simple. Tracks changes in the root set and merges filters with the same *forwards* sets. This requires management of the merged sets.

The merging mechanism requires that an additional data structure is used to keep track of merged nodes. The sets MR , CO , and optionally CV are needed for aggregate merging.

Inserting Filters. The insertion of a new filter f is only interesting when it is placed in the root set. If f is not mergeable, the *add* operation is performed. If f is covered by an existing filter or a merger, the corresponding *forwards* set is empty. For the *add* operation each new element f in the root set must be checked for covering by mergers. The new forwards set for f is

$$\text{forwards}'(f) = \text{forwards}(f) - \bigcup_{f' \in MR \wedge f' \sqsupseteq f} \text{forwards}(f'). \quad (1)$$

If f has a non-empty *forwards* set and is mergeable with an existing filter or filters, aggregate merging needs to be performed. Merging can be performed only for those filters that have the same *forwards* sets. Any mergers covered by a new root filter f are discarded if they have the same *forwards* sets. This approach may result in unnecessary updates if a merger covers another merger and they have differing *forwards* sets. On the other hand, this simplified approach does not require the complex tracking of the *forwards* sets.

Deleting Filters. Deletion of an existing filter f is only interesting if f is part of a merger or in the root set. When a filter that is part of a merger is removed, the merger is either re-evaluated if the size is greater than one, or removed if there is only one remaining filter in the merger. In either case the merger is unsubscribed. The corresponding uncovered set must be computed using the root set and forwarded with the unsubscription. The *forwards* sets of any direct successors to a removed merger must be re-evaluated. The *forwards* set is empty for any element in the successor set that is covered by other mergers.

4 Experimentation

A custom workload generator was used for the experimental results. The generator creates sets of filters and notifications using the given parameters. The key parameters of the workload generator are, the number of filters, the number of attribute filters in filters, the number of schemas, the number of unique names that are used in generating attribute filter constraints, the range of values for number tests, the number of notifications to match, and the number of interfaces. Interfaces are assigned to filters in a round-robin fashion.

Appendix A presents the perfect filter merging mechanism that we use in experimentation, but the techniques we presented in Section 3 are not restricted to this mechanism. The filters were generated using the structure enforced by a schema. Each attribute filter has a random type and a name with the restriction that the $\langle \text{name}, \text{type} \rangle$ pairs must be unique. Each attribute filter has a single predicate randomly selected from the set $\{<, >, \leq, \geq, =, \neq, [a, b]\}$.

Random and unique field names, schema names, and constraint names were generated using a uniform distribution over the set of lower-case alphabets with a uniformly distributed length of $U(3, 10)$. The range for integer values was 100. Notifications are generated as follows: each notification has the structure of the schema, integer tuples have a random value from the range [0, 100] and strings are drawn from the constraint name pool.

We used the following equipment: an HP laptop with a 2 GHz Pentium III and 512MB of main memory, Windows XP, and Java JDK 1.4.2. We used two different data structures, the filters poset and the balanced forest. The filters poset algorithm is based on the Siena Java implementation [15] and extended with hashtable-based duplicate filter detection.

Figure 2 presents an overview of experimentation with filter merging. The workload generator is used to generate filters and notifications. First the filter set is merged and then the resulting set is added to the underlying data structure. We record both merging time and insertion time.

Each interface-specific filter set is merged using the merging algorithm and the merging time is recorded. The merging time includes only the time spent in the merging of the root set for all interfaces and thus it represents the overhead of all neighboring routers and not the overhead of a single router. The average merging time for a single router can be obtained by dividing the merging time by the number of interfaces. The insertion time (add scenario) represents the time spent by a single router in processing the filter sets it has received.

The filters are also merged as a one-shot operation in the benchmark and the removal of a merged filter is not considered. The benchmark scenario corresponds to a situation in which a router receives already merged filter sets.

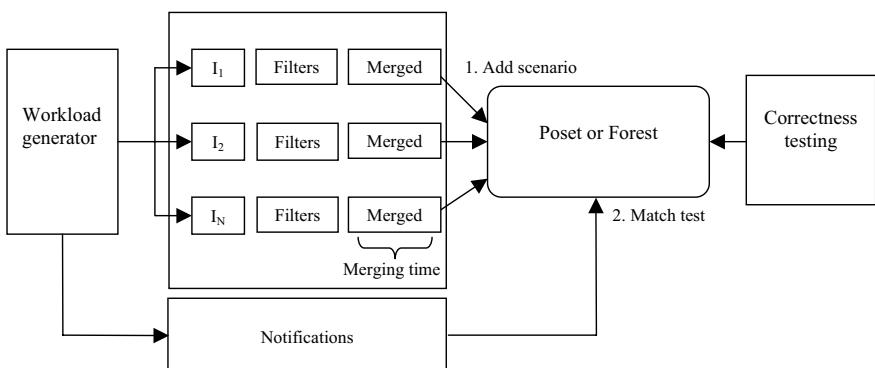


Fig. 2. Add scenario with merging

The two important cases in experimentation were a variable number of filters with unique interfaces and a variable number of interfaces with a static number of filters. We used a variable number of attribute filters (2-4) and 1 schema for the results. The motivation for using a variable number of attribute filters is that

it seems to be more realistic for multi-router environments than a static number of attribute filters, because user interests vary. Hence, we are using one attribute filter for the type and then 1-3 attribute filters for additional constraints. The single schema situation is the most difficult scenario for matching and merging, because two schemas are by definition independent and a notification may match only one schema (event type) at a time, but filters of the same schema have to be analyzed.

We measured matching time using a matching algorithm that walks only those sub-posets or sub-trees of the structure that match the notification. We compare this algorithm with a naive matcher that tests the notification against each filter.

4.1 Merging Results

Figure 3 presents the impact of interface specific merging for forest and poset performance with a static number of interfaces (3) and a variable number of filters. The merging benchmark compares the insertion and matching performance of interface-specific minimal cover sets with merged sets. 60 replications were used for these results. The merging time represents the worst case, because the input sets were merged using a one-shot procedure and normally this would be performed incrementally.

As the number of filters grows the merging algorithm is able to remove redundancy. The root size of the merged forest and the poset are the same and root

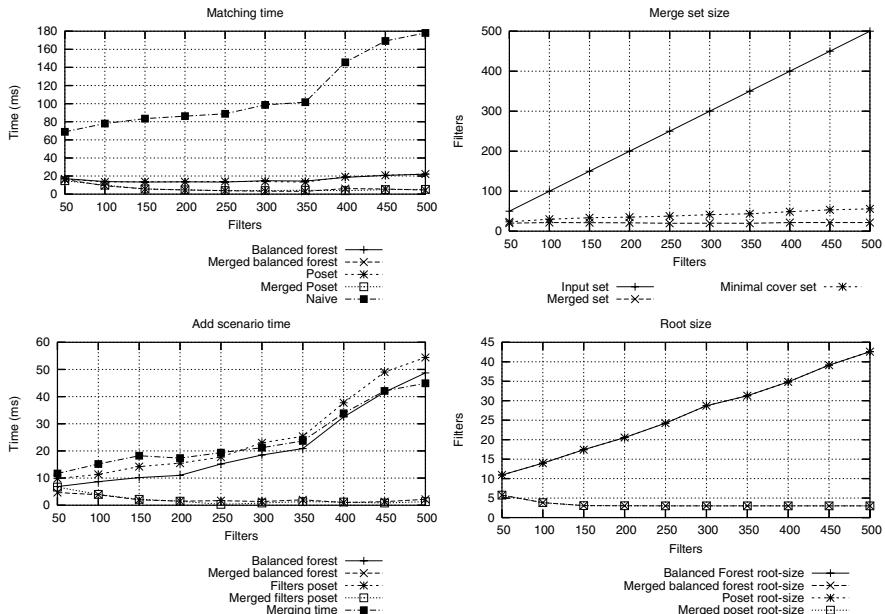


Fig. 3. Impact of merging on the forest and poset performance. Results for 3 interfaces

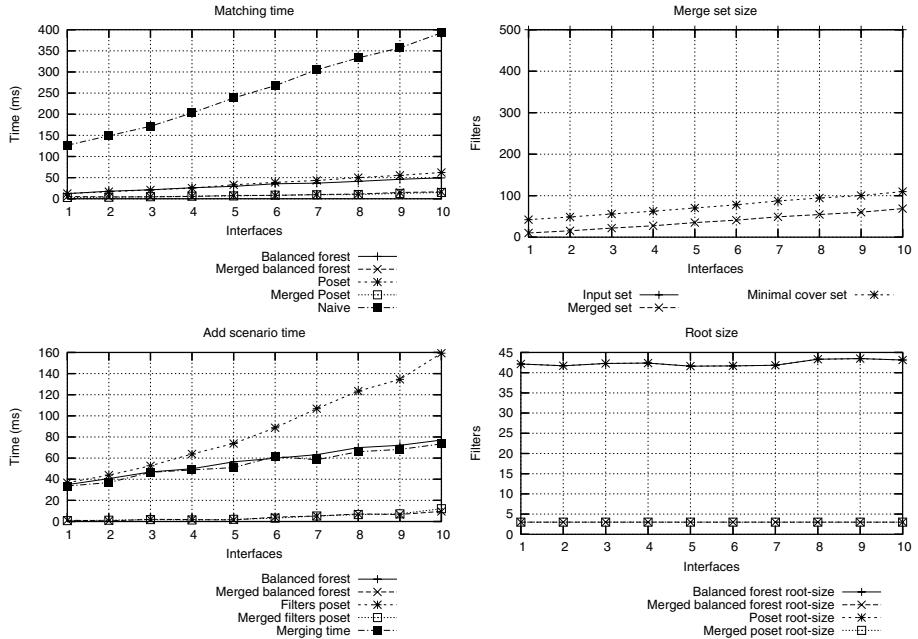


Fig. 4. Impact of merging on the forest and poset performance. Results for 500 filters.

set sizes in merging scenarios are considerably smaller than in normal operation. The root set sizes are shown for the merged and non-merged sets, respectively. The size of the non-merged set grows with the number of filters, whereas the size of the merged set is constant in the figure. Based on these results the merging time is reasonable and the merged forest or poset is created quickly. Matching time for the merged set is considerably shorter than for the non-merged set.

Figure 4 presents merging results for a variable interfaces and a static number of filters (500). The insertion and matching times for the merged sets are significantly lower also in this scenario. The processing performance decreases as the number of interfaces grows, because there are fewer filters per interface. The root sizes are very small for the merged sets whereas the non-merged sets are large. This is due to saturation, where the merged roots become very general when there are many filters.

4.2 Discussion

The results show that covering and merging are very useful and give significant reduction of the filter set, especially with variable number of attribute filters, because those filters with fewer attribute filters may cover other filters with more attribute filters. We have also experimented with a static scenario, where the number of attribute filters per filter is fixed. The static scenario gives also good results for covering, but perfect merging does not perform well when the number of attribute filters grows.

When the number of filters per schema grows the whole subscription space becomes covered, which we call *subscription saturation*. This motivates high precision filters for a small amount of filters, and more general filters when the subscription space becomes saturated.

The results indicate that filter merging is feasible and beneficial when used in conjunction with a data structure that returns the minimal covering set, and the filter set contains elements with a few attribute filters that cover more complex filters. The cost of unsubscription, or removing filters from the system, can be minimized by merging only elements in the minimal cover or root set. When a part of a merger is removed, the merger needs to be re-evaluated. Any delete or add operation outside the minimal cover does not require merging. For complex filter merging algorithms it is also possible to use a *lazy* strategy for deletions. In lazy operation, the system delays re-merging and counts the number of false positives. When a threshold is reached the minimal cover set or parts of it are merged and sent to relevant neighbors.

5 Conclusions

In this paper we presented a formal filter merging framework for content-based routers and other information processing applications, such as firewalls and auditing gateways. Filter merging is a technique to remove redundancy from filter sets. The proposed framework supports the merging of filters from local clients, slave routers in hierarchical operation, and neighboring routers in peer-to-peer operation. The compositionality of routing and filtering blocks is important for scalability and extensibility. Filter merging may be separately applied to various components of a router, namely the part that manages local clients and the part that handles external traffic between neighboring routers.

We discussed two ways to implement the formal framework, within the routing data structure and outside the data structure. We focused on the latter and the proposed aggregate merging mechanism is simple, independent of the used filtering language and routing data structure, and efficient. The system is efficient, because only the most general filters are considered to be candidates for merging. The system assumes that covering relations are known or can be computed for filters. We presented performance results for merging using current routing data structures for set-based merging. The results indicate that filter merging may be used to considerably reduce the processing overhead of neighboring routers in a distributed environment.

References

1. Betz, K.: A scalable stock web service. In: Proceedings of the 2000 International Conference on Parallel Processing, Workshop on Scalable Web Services, Toronto, Canada, IEEE Computer Society (2000) 145–150
2. Bharambe, A.R., Rao, S., Seshan, S.: Mercury: A scalable publish-subscribe system for Internet games. In: Proceedings of the 1st Workshop on Network and System Support for Games, Braunschweig, Germany, ACM Press (2002) 3–9

3. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems* **19** (2001) 332–383
4. Tarkoma, S., Kangasharju, J.: A data structure for content-based routing. In Hamza, M.H., ed.: Ninth IASTED International Conference on Internet and Multimedia Systems and Applications, ACTA Press (2005) 95–100
5. Mühl, G., Fiege, L., Gärtner, F.C., Buchmann, A.P.: Evaluating advanced routing algorithms for content-based publish/subscribe systems. In Boukerche, A., Das, S.K., Majumdar, S., eds.: The Tenth IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2002), Fort Worth, TX, USA, IEEE Press (2002) 167–176
6. Mühl, G.: Large-Scale Content-Based Publish/Subscribe Systems. PhD thesis, Darmstadt University of Technology (2002)
7. Carzaniga, A., Rutherford, M.J., Wolf, A.L.: A routing scheme for content-based networking. In: Proceedings of IEEE INFOCOM 2004, Hong Kong, China, IEEE (2004)
8. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.* **33** (2001) 322–373
9. Crespo, A., Buyukkokten, O., Garcia-Molina, H.: Query merging: Improving query subscription processing in a multicast environment. *IEEE Trans. on Knowledge and Data Engineering* (2003) 174–191
10. Wang, Y.M., Qiu, L., Achlioptas, D., Das, G., Larson, P., Wang, H.J.: Subscription partitioning and routing in content-based publish/subscribe networks. In D.Malkhi(Ed.), ed.: Distributed algorithms. Volume 2508/2002 of Lecture Notes in Computer Science. (2002)
11. Wang, Y.M., Qiu, L., Verbowski, C., Achlioptas, D., Das, G., Larson, P.: Summary-based routing for content-based event distribution networks. *SIGCOMM Comput. Commun. Rev.* **34** (2004) 59–74
12. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* **13** (1970) 422–426
13. Triantafillou, P., Economides, A.: Subscription summaries for scalability and efficiency in publish/subscribe systems. In Bacon, J., Fiege, L., Guerraoui, R., Jacobsen, A., Mühl, G., eds.: In Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02). (2002)
14. Triantafillou, P., Economides, A.A.: Subscription summarization: A new paradigm for efficient publish/subscribe systems. In: ICDCS, IEEE Computer Society (2004) 562–571
15. Department of Computer Science, University of Colorado: Siena Java language API and server code (2005)
16. Antollini, J., Antollini, M., Guerrero, P., Cilia, M.: Extending Rebeca to support concept-based addressing. In: First Argentine Symposium on Information Systems (ASIS 2004). (2004)
17. Mühl, G.: Generic constraints for content-based publish/subscribe systems. In Batini, C., Giunchiglia, F., Giorgini, P., Mecella, M., eds.: Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS'01). Volume 2172 of LNCS., Trento, Italy, Springer-Verlag (2001) 211–225

A Filter Merging Mechanism

This appendix presents the filter model and merging mechanism used in the experimentation.

A.1 Filter Model

A filter is represented with a set of attribute filters, which are 3-tuples defined by $\langle \text{name}, \text{type}, \text{filter clause} \rangle$. Name is an identifier and type is an element of the set of types. To simplify structural comparison of filters only conjunction may be used in defining the set of attribute filters. While disjunctions would give more power over the structure of the notification, they complicate the algorithms and, in any case, a disjunctive attribute filter set may be represented using a set of conjunctive attribute filter sets. The filter clause defines the constraints imposed by the attribute filter to a corresponding tuple in a notification identified by the name and type.

The attribute filter consists of atomic predicates and it may have various semantics. The simplest attribute filter contains only a single predicate. This kind of format is being used in many event systems, such as Siena and Rebeca. A more complex format supports conjuncts and disjuncts, but they complicate filter operations such as covering, overlapping, and merging.

We define the filter clause to support atomic predicates and disjunctions. In general, covering and overlapping of arbitrary filters in the disjunctive normal form may be determined using expression satisfiability. Expressions written in the disjunctive normal form are satisfiable if and only if at least one of those disjunctive terms is satisfiable. A term is satisfiable if there are no contradictions [16].

A.2 Covering

Covering relations exist between four different components: predicates, disjuncts, attribute filters, and filters. Filter covering is an important part of the framework and it is used by the poset-derived forest data structure to determine the relationship between filters and the merging algorithm to remove redundancy from attribute filters.

The covering test for an attribute filter is similar to the covering test for filters and conjunctive formulas presented in [6]. Theorem 1 presents covering for disjunctive attribute filters. A pre-requirement is that filter A has the same name and type than B . The other direction requires a more complicated mechanism and proof. It is envisaged that this implication is still useful for simple and efficient operation. By using Theorem 1 it is possible that not all relations are captured, but if used in a consistent manner it does not alter routing semantics and provides an efficient way to compute covering relations.

Theorem 1. *Let $A = \bigvee_{j=1}^m A_j$ and $B = \bigvee_{i=1}^n B_i$, where the A_j 's and B_i 's are predicates. If $\forall i \exists j A_j \supseteq B_i$ then $A \supseteq B$.*

Proof. Assume $\forall i \exists j A_j \sqsupseteq B_i$. Then, when B is true, some B_i is true. But by assumption there then exists some true A_j , which means that A is true. By definition of the covering relation we then have $A \sqsupseteq B$.

A.3 Attribute Filter Merging

Attribute filter merging is based on two mechanisms: covering relations between disjuncts and perfect merging rules for atomic predicates. Covering relations are used to remove any covered disjuncts. Perfect merging rules are divided into two categories: existence tests and predicate modification. The former combines two input predicates into an existence test when the merging condition is satisfied. The latter combines two predicates into a single predicate when the condition is satisfied. A merged disjunct may cover other disjuncts, which requires that the other disjuncts are checked for covering after a merge.

We assume that the length of a disjunct is not important — if a cost function is associated with the selection of the disjuncts to be merged and covered, the computational complexity of merging may not necessarily be polynomial.

The merging rules for the number existence test are presented in Table 1. The existence test conditions are given for two input filters, F_1 and F_2 . The condition must be satisfied in order for the two input filters to merge to an existence test. Each filter has a predicate and an associated constant denoted by a_1 and a_2 , respectively. Ranges are denoted by two constants a and b . For example, given $x < a_1$ and $x \neq a_2$ where $a_1 = 10$ and $a_2 = 7$ the condition $a_1 > a_2$ is satisfied and results in an existence test. If a_2 is greater or equal to a_1 the condition is no longer satisfied.

Table 1. The rules for number existence test

F_1	F_2	Condition	F_1	F_2	Condition
$x = a_1$	$x \neq a_2$	$a_1 = a_2$	$x \neq a_1$	$x = a_2$	$a_1 = a_2$
$x < a_1$	$x > a_2$	$a_1 > a_2$	$x < a_1$	$x \geq a_2$	$a_1 \geq a_2$
$x \leq a_1$	$x > a_2$	$a_1 \geq a_2$	$x \leq a_1$	$x \geq a_2$	$a_1 \geq a_2$
$x > a_1$	$x < a_2$	$a_1 < a_2$	$x > a_1$	$x \leq a_2$	$a_1 \leq a_2$
$x \geq a_1$	$x < a_2$	$a_1 \leq a_2$	$x \geq a_1$	$x \leq a_2$	$a_1 \leq a_2$
$x \neq a_1$	$x \neq a_2$	$a_1 \neq a_2$	$x \neq a_1$	$x > a_2$	$a_1 > a_2$
$x \neq a_1$	$x \geq a_2$	$a_1 \geq a_2$	$x \neq a_1$	$x \leq a_2$	$a_1 \leq a_2$
$x \neq a_1$	$x < a_2$	$a_1 < a_2$	$x > a_1$	$x \neq a_2$	$a_1 < a_2$
$x \geq a_1$	$x \neq a_2$	$a_1 \leq a_2$	$x < a_1$	$x \neq a_2$	$a_1 > a_2$
$x \leq a_1$	$x \neq a_2$	$a_1 \geq a_2$	$x \neq a_1$	$x \in [a, b]$	$a_1 \in [a, b]$
$x \in [a, b]$	$x \neq a_1$	$a_1 \in [a, b]$			

A.4 Filter Merging

The perfect merging algorithm merges two filters that have identical attribute filters except for one pair of distinctive attribute filters. These distinctive attribute

filters are then merged. For disjunctive attribute filters the distinctive attribute filters are always mergeable. Conjunctive attribute filters are not necessarily mergeable. For any two mergeable filters F_1 and F_2 the operation $\text{merge}(F_1, F_2)$ either merges the distinctive attribute filters or the filters are identical.

This type of merging is called perfect, because covering and perfect merging rules do not lose or add information. More formally, a merger M of filters $\{F_1, \dots, F_n\}$ is perfect if and only if $N(M) = \bigcup_i N(F_i)$. Otherwise, the merger is called imperfect [6, 16].

The selection of the best merging candidate is an important part of the merging algorithm. First, the best filter must be located for merging. Second, the best attribute filter or filters within that filter must be selected for merging. Our current implementation selects the first mergeable candidate; however, a more optimal merging algorithm would select the candidate that has the most general merging result, because in some cases merging may add complexity to a filter in the form of a disjunct. This latter behaviour reduces a filter's probability to merge with other filters in the future. Therefore a good candidate filter for merging is one that either has less predicates or disjuncts after the merge operation or the predicates and disjuncts are more general. In the best case, the merged filter will cover many previously uncovered filters.

A.5 Discussion

The single predicate and set-based perfect merging approach presented in [6, 17] requires that all attribute filters are identical except for one pair of distinctive attribute filters. It is not always possible to merge simple constraints: for example the perfect merging of two ranges $[0, 20]$ and $[30, 40]$ is not possible using conjunctive or single predicate attribute filters. The presented approach is more expressive, because it supports disjunctions.

Don't Mind Your Vocabulary: Data Sharing Across Heterogeneous Peers

Md. Mehedi Masud¹, Iluju Kiringa¹, and Anastasios Kementsietsidis²

¹ SITE, University of Ottawa

{mmasud, kiringa}@site.uottawa.ca

² School of Informatics, University of Edinburgh

akements@inf.ed.ac.uk

Abstract. The strong dynamics of peer-to-peer networks, coupled with the diversity of peer vocabularies, makes query processing in peer database systems a very challenging task. In this paper, we propose a framework for translating expressive relational queries across heterogeneous peer databases. Our framework avoids an integrated global schema or centralized structures common to the involved peers. The cornerstone of our approach is the use of both syntax and instance level schema mappings that each peer constructs and shares with other peers. Based on this user provided mapping information, our algorithm applies generic translation rules to translate SQL queries. Our approach supports both query translation and propagation among the peers preserving the autonomy of individual peers. The proposal combines both syntax and instance level mappings into a more general framework for query translation across heterogeneous boundaries. We have developed a prototype as a query service layer wrapped around a basic service providing heterogeneity management. The prototype has been evaluated on a small peer-to-peer network to demonstrate the viability of the approach.

1 Introduction

In the past few years, Peer-to-Peer (P2P) applications have emerged as a popular way of sharing data in decentralized and distributed environments. In such environments, involved data sources, called peers, act autonomously in their sharing of data and services. A peer database system (PDBS) consists of a peer or node of a P2P network which has been augmented both with a conventional data base management system and an interoperability layer that enables data sharing across (usually) heterogeneous boundaries. The local databases on each peers are called *peer databases*. Each PDBS is independent of others and maintains its own peer databases. In addition to the latter, each PDBS needs to establish value correspondences between its local data and data on remote PDBSs for the purpose of data sharing. Such value correspondences constitute logical links that we call *acquaintances*. For example, we may consider a network of peer database systems of family doctors, hospitals, medical laboratories, and pharmacists that are willing to share information about treatments, medications and

test results of their patients. Consider a situation where a patient has an accident in a city where he is currently visiting. He then visits a walk-in clinic. The doctor in the clinic needs to know the patient's previous medications and treatments from the patient's family physician. The associated doctor in the walk-in clinic establishes acquaintances with the patient's family doctor and pharmacist for sharing information about the patient. The doctor in the walk-in clinic then retrieves information from the acquainted peers, for instance family physicians and medical laboratories. In this perspective, we need particularly fine-grained data coordination, sharing and query processing strategies. Coordination rules need to be dynamic because peers join and leave the system freely. In such a dynamic environment, the existence of a global schema for the entire databases is not feasible [4].

Query processing techniques in peer database networks can be compared to similar techniques used in the traditional distributed and multi database systems which use global or mediated schema to allow viewing of all involved databases as one single database.

On the contrary, in general, peer database systems have no such global schemas and permit true autonomy to peers. In addition, queries are posed on local database and results are retrieved from the local database as well as from the peer database network. In this paper, we make the following contributions.

- First, we present a query translation mechanism that does not use a restrictive global/mediated schema and that considers the heterogeneity and autonomy of peer databases. The algorithm translates arbitrary Select-Project-Join SQL queries, where the selection formula may contain both conjunctive and disjunctive normal form with negation.
- Second, the algorithm translates queries according to a collection of user-generated rules that combine a restricted form of syntactic schema mappings with the data instance level mappings. We also present a mapping stack consisting of four layers of mappings that each peer uses partially or fully in order to create semantic relationships with other peers.
- Finally, we implemented our query translation algorithm on top of the increasingly reliable P2P JXTA framework [1]. We ran the implementation on six JXTA-based peers. We show measurements that indicate that the algorithm is efficient.

The remainder of the paper is organized as follows. Section 2 describes a usage scenario. In Section 3, we discuss peer-to-peer mapping semantics, followed by Section 4 that discusses the peer-to-peer query semantics. Section 5 presents the query translation rules and framework and Section 6 describes the query translation algorithm. In Section 7, we show experimental setup and results. Section 8 treats related work.

Finally, we address the scope for future work and conclude in Section 9.

2 Motivating Example

We use a motivating example from the Health Care domain where hospitals, family doctors and pharmacies, all share information about patients. This information includes health histories, medications, and exams. Figure 1 depicts the schemas used for this domain. The Family Doctor peer has a unique Ontario Health Insurance Patient (OHIP) number assigned to each patient and record is kept of name, illness, date of visit and medication of patients. The relation *MedicalExam* stores the information about the patients' medical examinations. The Hospital peer has relations *Patients* and *Labtest*. Table 2 shows partial instances of both peer databases. Assume that a patient has been admitted to a hospital. The doctor in the hospital needs the medical examination that the patient has gone through on admission as well as the patient's recent test reports from the patient's family doctor. To get the medical examination, the doctor in the hospital may pose the following query against the local Hospital peer database:

*Q1: select * from LabTest where PATID=“243” AND Test=“C0518015”*

To get the test reports, the very same doctor needs to have the following query posed against the remote peer database of the patient's family doctor:

*Q2: select * from MedicalExam where OHIP=“501NE” AND TestName=“homoglobin”*

Normally, due to the autonomy of peer databases, the doctor in the hospital should be able to pose the later query in terms of the schema and instance data values of his local peer database. However, Figure 2 shows that there are differences in the way patients' information is represented both at the schema and at the data instance level in the two peer databases. This representational discrepancy raises the need for some way of mapping one representation to the other, both at the schema and at the data instance levels. Such a mapping will permit interpretability between these heterogeneous scenarios for query translation. We will use mapping tables [15] to resolve heterogeneity at the data level and syntactic mappings of schema element for schema level heterogeneity during

Patient (OHIP, Lname, Fname, Illness, Date)
 MedicalExam(OHIP, TestName, Result)

(a) Family doctor database

Patients (PATID, Primary_Desc, Name)
 Labtest (TestID, Test, Result, PATID)

(b) Hospital database

Fig. 1. Database schemas

OHIP	Lname	Fname	Illness	Date
233GA	Lucas	Andrew	Headache	Jan/04
501NE	Davidson	Ley	Allergy	Jan/04

(a) Patient table instance

OHIP	TestName	Result
233GA	whitebloodcount	9755 c/mcL
501NE	homoglobin	14.6 g/dL

(b) MedicalExam table instance

PATID	Name	Primary_Desc
243	Davidson,ley	StomachPain
359	Lucas, Andrew	Heart Problem

(c) Patients

TestId	Test	Result	PATID
4520	C0427512	633 c/mcL	359
4521	C0518015	12.5 g/dL	243

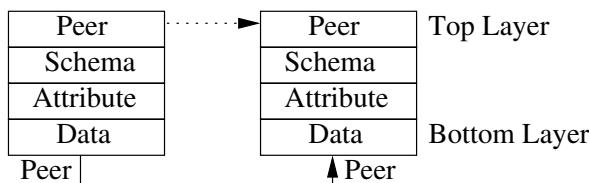
(d) LabTest table instances of hospital database

Fig. 2. Database instances

query translation. Based on these mappings, we develop several generic query translation rules that translate a query q posed on a peer P to a set of queries $Q' = \{q_1, \dots, q_n\}$, where each one of the q_i 's is the original query q translated to the vocabulary of a given acquainted peer P_i .

3 Peer-to-Peer Mappings

In what follows, we assume that sources are fully autonomous and may partially or fully share information at different levels. We consider pairwise mappings between peers with their shared schema elements. A mapping consists of four layers. Figure 3 shows the four layers of mappings. The top layer is the peer-to-peer mapping, which defines the acquaintance between two peers. Schema elements are exchanged between peers as part of the acquaintance protocol. The second layer is the schema level mapping which helps to overcome schema level heterogeneity. The next layer is the attribute level mapping that is used to overcome heterogeneity at the attribute level. The last one is called instance/data level mapping. We use this layer if there are any differences in data vocabularies between attributes of two peer relations. We use the concept of mapping tables to create the data layer mapping.

**Fig. 3.** Peer to Peer mapping stack

There may be four different types of mappings between schema elements. They are as follows: one to one, one to many, many to one, and many to many. To take on one these, in a one to many mapping, a schema element such as an attribute/relation of one peer is mapped to more than one attribute/relation of another peer. For example the attribute *Name* of relation *Patients* is mapped to attributes *Lname* and *Fname* of relation *Patient*. Moreover, a set of attributes of one relation in one peer may be mapped to more than one relation in another peer.

3.1 Mapping Tables

Mapping tables [15] are used to represent the data value correspondences between attributes of two different relations to account for differences in data vocabulary. Intuitively, a mapping table is a relation over the attributes $X \cup Y$, where X and Y are non-empty sets of attributes from two peers. For example, Figure 4 shows a mapping table representing a mapping from the set of attributes $X=\{OHIP\}$ to the set of attributes $Y=\{PATID\}$. The same table shows another mapping table that relates *MedicalExam.TestName* and *LabTest.Test*.

Mapping tables represent expert knowledge and are typically created by domain specialists. Currently the creation of mapping tables is a time-consuming and manual process performed by a set of expert curators. Still there is no complete automated tool to facilitate the creation, maintenance and management of these tables [15]. However, the paper [15] introduces two mechanisms: i. *Infer new mapping tables* to find a set of all mapping tables that are valid and available over a network of peers and ii. *Determine consistency of mapping table* to ensure consistency when update occurs in mapping tables. However, both of these mechanisms play an important role in helping a curator understand and correctly specify the semantics of a set of mapping tables.

OHIP	PATID	TestName	Test
501NE	243	homoglobin	C0518015
233GA	388	whitebloodcount	C0427512

(a) Mapping table OHIP2PATID (b) Mapping table Test-Name2Test

Fig. 4. Example of mapping tables

We can treat mapping tables as constraints in exchanging information between peer databases [15]. In the present paper, we use mapping table in this sense. We assume a closed world semantics for mapping tables.

3.2 Data and P2P Network Model

This section introduces some basic notions that will be used throughout the paper. A *database schema* is any nonempty, finite set $DB[W]=\{R_1[U_1], \dots, R_n[U_n]\}$

of relations, where U_i is a subset W , the set all available attributes, and R_i is a relation name; $R_i[U_i]$ denotes a relation R_i over a set U_i of attributes. Given a relation R , and a query q , we will use the notation $att(R)$ and $att(q)$ to denote the set of attributes mentioned in R and q respectively. *Instances* of a relation schema $R_i[U_i]$ and a relational database $DB[W]$ are defined in the usual way.

Now we formally introduce the notion of a network of PDBSs.

Definition 1. A network of PDBSs is a pair $(\mathcal{N}, \mathcal{M})$. Here, $\mathcal{N} = \{(\mathcal{P}, \mathcal{L})\}$ is an undirected graph, where $\mathcal{P} = \{P_1, \dots, P_n\}$ is a set of peers, $\mathcal{L} = \{(P_i, P_j) | P_i, P_j \in \mathcal{P}\}$ is a set of acquaintances. Each peer P_i is associated with an instantiated relational database – a peer database – with schema $DB_i[W_i]$, and each acquaintance (i, j) is associated with a set $\mathcal{M}_{ij} \in \mathcal{M}$ of mapping tables.

We will interchangeably call networks of PDBSs “P2P networks” that peers make only a subset of its schema visible to its peers. That is, we assume that each peer P_i exports a (possibly empty) subset $V_i \subseteq DB_i[W_i]$ of schema elements (i.e. attributes or relations) called *export schema* of P_i . For simplicity, we assume that $V_i \cap V_j = \emptyset$, for all $i \neq j$.

4 Peer-to-Peer Query Semantics

We assume that each query in PDBSs is defined in terms of the schema of a single peer. Each user is only aware of the local database schema. We also assume unrestricted select-project-join SQL queries. There are two types of queries in PDBs, namely local and global queries [8]. A local query is defined in terms of a local peer database schema, while a global query is defined over the peer database schema of the P2P network that are directly or indirectly acquainted with the peer where the global query is initiated. Semantically, a global query is a set of queries translated from the local query, all destined to acquainted peers. A global query is generated when a user wants results from all reachable (directly or indirectly) acquainted peers in the P2P network.

This informal semantics of queries can be formalized as follows (slightly modifying the semantics given in [14]). Suppose a network $\mathfrak{N} = (\mathcal{N}, \mathcal{M})$ of PDBSs, where $\mathcal{N} = (\mathcal{P}, \mathcal{L})$ and $\mathcal{P} = \{P_1, \dots, P_n\}$. A local query q in a peer $P_i \in \mathcal{P}$ is defined over (a subset of) the schema $DB_i[W_i]$ of P . The answer to q is a relation over the instance db_i of $DB_i[W_i]$. A global query $q_{\mathfrak{N}}$ over the P2P network \mathfrak{N} is a set $\{q_1, \dots, q_k\}$ ($1 \leq k \leq n$), where each query q_i ($1 \leq i \leq k$) is a *component query* defined over the schema of a reachable peer. The intuition behind this definition is the following: each component query q_i is a user defined query that has been forwarded to all reachable peers via translation through the given mapping tables. The answer to the global query $q_{\mathfrak{N}}$ is the set $\{q_1(db_{i_1}), \dots, q_k(db_{i_k})\}$, where $q_j(db_{i_j})$ ($1 \leq j \leq k$) is a relation over the instance db_{i_j} of peer P_j .

For query propagation we use a *translation – and – forward* mechanism. When a user poses a query on a peer then the peer first translates the query for all acquainted peers and sends the translated queries to its acquaintances. Before

sending the translated query the peer first adds a tag, we say global identification (GID) of the query as well as the peer identification (PID). When a remote peer receives the query, the peer either translates and/or forwards the query adding its PID with the query. This *translation – and – forward* process continues until no further propagation is possible. The global identification (GID) is used to avoid duplicate translation of a query in a peer because a peer may receive queries in different form but with same GID from multiple acquaintances. Therefore, if a peer receives a query with the same GID as a query already seen, then the new query is rejected. The path tag makes this possible and thus helps avoid cycles. So the path tag is used to trace from which peer the query has been originated and the list of peers the query has been visited. Therefore, looking at the path tag, a peer knows whom to forward the query.

5 Query Translation

Query translation is the problem of transforming the query vocabulary of q over the schema of a local peer P to a query q' over the schema of an acquainted peer P' . The query vocabulary refers to three types of information, namely the set of *attributes* that represent the answer or result of the query, the *data values* mentioned in query search conditions, and the *relation names* mentioned in the query.

Example 1. Consider the following query which retrieves OHIP number and test result of a patient with $Lname = “Lucas”$, $Fname = “Andrew”$ and $TestName = “whitebloodcount”$.

Q3: select OHIP, Result, Date from Patient, MedicalExam

```
where Patients.OHIP=MedicalExam.OHIP AND (Lname=“Lucas”
AND Fname=“Andrew”) AND Test=“whitebloodcount”
```

From the above query we find the following query vocabularies: i. the set of attributes represent the answer or result of the query:(OHIP, Result, Date) ii. the set of query search conditions ($Lname=“Lucas”$, $Fname=“Andrew”$, and $Test=“whitebloodcount”$) and iii. the set of relations (Patient, MedicalExam).

In order to pose the query in terms of vocabularies of acquainted peers for example the peer *Hospital*, the translated query should be as follows.

Q4: select PATID, Result from Patients, LabTest

```
where Patients.PATID=LabTest.PATID AND Name=“Lucas, Andrew”
AND Test=“C0427512”
```

In order to translate the query vocabularies that means query attributes, relations and data vocabularies in search conditions we use the notion of correspondence assertions and introduce translation rules. We simply translate the query attributes and relations from correspondence assertions and search conditions from translation rules. The semantics of correspondence assertion and translation rules are defined in the following two sections. We later show how these two things are used to translate queries.

5.1 Correspondence Assertion (CA)

In our peer to peer system we assume that each peer exports part of their schema as a shared schema that are made available to the system to create acquaintances with other peers. Mapping tables are also placed in peers to resolve difference in data vocabulary with acquainted peers. Therefore, we need to formally characterize the relationship between exported elements (attributes/relations) between acquainted peer schemas. We capture this relationship in the notion *correspondence assertion* (CA) between peers' shared schemas. We can also create correspondence assertions between two attributes that form a mapping table. In general, a mapping table $m[X \cup Y]$ encodes, in addition to the set of data associations, an attribute correspondence between the set of attributes X and Y . This generation of CAs occurs at acquaintance time.

Example 2. Consider the instances in Figure 2 and the mapping tables in Figure 4. Suppose the Family Doctor peer has the following export schema:

$$V = \{OHIP, Lname, Fname, TestName, Result\}$$

Also suppose the Hospital peer has the following export schema:

$$V = \{patid, name, test, result\}$$

Therefore we create the following correspondence assertions.

CA1: OHIP → PATID, CA2: Name, Fname → Name, CA3: TestName → Test

5.2 Translation Rule

In this section we introduce some query translation rules that are used to translate query search conditions and their data vocabularies. For example consider the query Q3 and Q4. In the query Q3, the search condition for patient name is given using ($Lname = "Lucas"$, $Fname = "Andrew"$) and test name is given using ($Test = "whitebloodcount"$). But the patient name is represented in peer *Hospital* is represented in different format and the condition should be translated as $name = 'Lucas, Andrew'$ to be recognized in peer *Hospital*. Also the test name “whitebloodcount” is represented in peer *Hospital* with different data vocabulary “C0427512”. For this purpose we need some kind of translation rules to resolve these heterogeneity. In this paper we address four translation rules are named as Merge (M), Separation (S), Data Association (DA), and Data Conversion (DC). Each one of the following sections describes each one of these query translation rules and shows how these rules translate query search conditions.

Merge Rules (M). This rule resolves differences in format. Therefore, we need a mechanism to translate the data format in the selection formula of the source query to the format of the formula that represents the target selection formula. We represent a merge rule as follows:

$$\sigma_{\bigwedge A_i=x_i} : \sigma_{B=y} : y = \Pi_B(R' \bowtie T_{\bigwedge A_i=x_i}), \quad (1)$$

where $R' = f_M(R(A_1, \dots, A_n), B)$; $\sigma_{\wedge A_i=x_i}$ is a pattern of a selection formula in the source query and $\sigma_{B_i=y_i}$ is the translated selection formula for the target query. The value of y for attribute B is determined by the formula defined in the translation expression of the rule. The semantics of formula above is as follows:

- $f_M(R(A_1, \dots, A_n), B)$ is a function that creates a temporary relation R' from relation R with attributes A_1, \dots, A_n , and B ; A_1, \dots, A_n are mentioned in $\sigma_{\wedge A_i=x_i}$ and B is the target attribute. Values of attribute B are generated with a user-provided function f that is applied on attributes A_1, A_2, \dots, A_n .
- $T_{\wedge A_i=x_i}$ is a tabular representation of the term $\wedge A_i = x_i$.

Separation Rule (S). The separation rule is the reverse of the merge rule. The formal representation of the separation rule is:

$$\sigma_{A=x} : \sigma_{\wedge B_i=y_i} : y_i = \Pi_{B_i}(R' \bowtie T_{A=x}), \quad (2)$$

where $R' = f_S(R(A), (B_1, \dots, B_n))$; $\sigma_{A=x}$ is a pattern of a selection formula and $\sigma_{\wedge B_i=y_i}$ is the translated selection formula. The value of y'_i s for attribute B'_i s are determined by the formula defined in the translation expression of the rule. The semantics of formula above is as follows:

- $f_S(R(A), (B_1, \dots, B_n))$ is a function that creates a temporary relation R' from relation R with attributes A and B_1, \dots, B_n ; A is mentioned in $\sigma_{\wedge A=x}$ and B'_i s are the target attributes for the translated query. Values of attribute B'_i s are generated with a user-provided function f that is applied on attributes A .
- $T_{A=x}$ is a tabular representation of the term $A = x$.

Data Association Rule (DA). Our third translation rule translates queries based on mapping tables. This rule deals with data level heterogeneity. When a query mentions a data value that differs from data values used in an acquainted peer, then we need to translate the predicate term in such a way that other peers are able to decipher it. We use mapping tables to translate this type of predicate term. The formal representation of the rule is:

$$\sigma_{\wedge A_i=x_i} : \sigma_{\vee B_j=y_j} : y_j = \Pi_B(m(A, B) \bowtie T_{\wedge A_i=x_i}), \quad (3)$$

where $\sigma_{\wedge A_i=x_i}$ is a pattern of a selection formula in a local query q and $\sigma_{\vee B_j=y_j}$ is the translated selection formula for target queries, and A and B are sets of attributes. The value of y_j for attribute B_j is determined by the formula $\Pi_B(m(A, B) \bowtie T_{\wedge A_i=x_i})$. Here, attributes A_i are those mentioned in $\sigma_{\wedge A_i=x_i}$.

Data Conversion (DC). This rule is used for translating data from one domain to another. For example, consider an attribute *height – in – inches* from one database and an attribute *height – in – centimeters* from another.

The value correspondence of these two attributes can be constructed by defining a conversion function f . Formally we represent this rule as follows:

$$\sigma_{A \text{ op } x} : \sigma_{B \text{ op } y} : y = \Pi_B(R' \bowtie T_{A \text{ op } x}), \quad (4)$$

where $R' = f_{DC}(R(A), B)$; $\sigma_{A \text{ op } x}$ is a pattern of a selection formula in a local query q and $\sigma_{B \text{ op } y}$ is the translated selection formula for the target query; f_{DC} is a user-defined function for data conversion; and $T_{A \text{ op } x}$ is the tabular form of $A \text{ op } x$. If the domain of attribute A and B are same, then function f_{DC} is called an identity function.

6 Query Translation Process

We start our query translation process defining the relationship between correspondence assertions and translation rules. We define the relationship between a correspondence assertion and a translation rule as a pair (CA, R) . We use the following syntax to represent the relationship between CA and translation rule R.

$$A \longrightarrow B : r$$

Where $A \subset V$ and $B \subset V'$. V and V' are the exported schemas from peer P and P' . Each rule $r \in \mathcal{R}$ defined in section 5.2 describes the relationship between attributes of correspondence assertions and how the vocabularies are translated in terms of structural, format, and data values. The translation process starts when a user poses a query on its local database and wants a global execution of the query. query to its acquaintances. The algorithm to translate queries is shown in Figure 5. The algorithm mainly translates the selection part of a query. There are two steps for translating a selection of a query. Firstly, the query is analyzed syntactically and the query condition is transformed into a *query condition tree* which is defined as follows.

Definition 2. Suppose q is a query. The Query Condition Tree (QCT) of q is a tree representing the selection formula of q , where the inner nodes are boolean operators AND, OR, or NOT, and leaves are atomic conditions of the form $A \text{ op } x$, where A is an attribute, op is a comparison operator $<$, $>$, $=$, \leq , \geq , or \neq , and x is a constant value.

Secondly, the query condition part is decomposed in terms of the correspondence assertions. The function *FindPartition* performs this task. The function *FindPartition* is shown in Figure 6. The algorithm takes as input a query tree and predefined set of correspondence assertions. Its output is a set $P = \{P_1, \dots, P_n\}$, where each P_i is a triple (T, CA, R) ; T is a set of predicate terms that are resulted from partitioning the original query predicate terms; CA is the corresponding correspondence assertion that maps the terms in T , and R is the corresponding translation rule that will be used to translate the predicate terms in T . The function *FindPartition* finds the potential partitions that

Query Translation (Q)**Input:** A query Q from user.**Output:** Translated query Q'

```

begin  $QCT =$  Create AND-OR-NOT tree from the
      query conditions;
 $CM =$  FindPartition( $QCT$ );
      /*CM: Set of matchings for query conditions */
for each  $cm_i \in CM$  do
    Apply rule  $r_i$  associated with  $cm_i$ ;
    Translate the constraint using translation rules;
    Translate ( $cm_i$ );
end for
end

```

Fig. 5. Query translation main procedure

can be translated independently. That means partitions are disjoint and there is no dependency between the terms in the partitions. Consider the following complex query:

Q5: select lname, fname, result from Patient,MedicalExam
 where Patient.OHIP=MedicalExam.OHIP AND
 (((lname=“Hall” OR lname=“Hull”) AND (fname=“Andrew”)) OR
 (OHIP=“233GA”)) AND (TestName=“whitebloodcount” AND Date=“Jan/04”)

where the predicate P is as follows:

((((lname=“Hall” OR lname=“Hull”)AND(fname=“Andrew”))OR(OHIP=“233GA”))
 AND (TestName=“whitebloodcount” AND Date=“Jan/04”))

Consider that we have following associations between correspondence assertions and rules:

ca1: $lname \rightarrow name : 4$, ca2: $lname, fname \rightarrow name : 1$
 ca3: $OHIP \rightarrow PAITD : 3$, ca4: $TestName \rightarrow Test : 3$
 ca5: $Date \rightarrow Dt : 4$

We assume that correspondence assertions ca1, ca5 are bound to rule 4(Data Conversion), ca2 is bound to rule 1(Merge Rule) and ca3, ca4 are bound to rule 3(Data Association). Finding potential partitions is important, because the composition of two terms in a query predicate may map to a particular correspondence assertion. Also sometimes, it is possible that a term can not be translated independently but only in combination with another term can the composed term be mapped with a correspondence assertion. The lines 2-9 of the algorithm *FindPartition* first find the mappings based on correspondence assertions. At this point we find the following initial mappings for our example:

FindPartition(CTree, CA)

Input: A query condition tree CTree and set of Correspondence Assertions CA.

Output: Potential partitions $P = \{P_1, \dots, P_n\}$ of constraints begin

```

1.   CM = /* CM is a set of pair  $(t, ca_i)$  where  $t$  is a atomic term
      and  $ca_i$  is corresponding correspondence assertion that covers attribute of t
2.   for each term  $t$  at leaf,  $t \in T$  /*  $T$  is a set of predicate terms in CTree*/
3.       M = {} /* M is a set of matches found in CA for term  $t$  */
4.       for each correspondence assertion  $ca_i \in CA$ 
5.           if attribute( $c$ )  $\in$  attribute( $ca_i$ ) then
6.               M = M  $\cup$   $ca_i$ 
7.               CM = CM  $\cup$   $(t, M)$ 
8.           end for
9.       end for
10.      for each  $m_i \in CM$ 
11.           $m_k = \emptyset$ 
12.          for each  $m_j \in CM$  and  $m_i \neq m_j$ 
13.              if  $(m_j(ca) \cap m_i(ca)) \neq \emptyset$  then
14.                   $m_k(t, ca) = \{\{m_i(t) \cup m_j(t)\}, \{m_i(ca) \cap m_j(ca)\}\}$ 
15.                  P = P  $\cup$   $m_k(t, ca)$  /*Forming partition*/
16.                  CM = CM -  $m_j$ 
17.              end for
18.              if ( $m_k = \emptyset$ )
19.                  P = P  $\cup$   $m_i(t, ca)$ 
20.                  CM = CM -  $m_i(t, ca)$ 
21.              end for
22.          end for
23.          return P
end

```

Fig. 6. Finding partition

lname = “Hall”, $M = [lname \rightarrow name, lname, fname \rightarrow name]$
 $CM1 = [lname = “Hall”, [lname \rightarrow name, lname, fname \rightarrow name], 4]$
lname = “Hull”, $M = [lname \rightarrow name, lname, fname \rightarrow name]$
 $CM2 = [lname = “Hull”, [lname \rightarrow name, lname, fname \rightarrow name], 4]$
fname = “Andrew”, $M = [lname, fname \rightarrow name]$
 $CM3 = [fname = “Andrew”, [lname \rightarrow name, lname, fname \rightarrow name], 1]$
OHIP=“233GA”, $M = [OHIP \rightarrow PATID]$
 $CM4 = [OHIP = “233GA”, [OHIP \rightarrow PATID], 3]$
TestName=“whitebloodcount”, $M = [TestName \rightarrow Test]$
 $CM5 = [TestName = “whitebloodcount”, [TestName \rightarrow Test], 3]$
Date=“Jan/04”, $M = [Date \rightarrow Dt]$
 $CM6 = [Date = “Jan/04”, [Date \rightarrow Dt], 4]$

The lines 10-22 perform the task of finding potential partitions from the above mappings. Therefore we get the following partitions.

$P1 = [lname = “Hall”, fname = “Andrew”, [lname, fname \rightarrow name], 1]$
 $P2 = [lname = “Hull”, fname = “Andrew”, [lname, fname \rightarrow name], 1]$

$$\begin{aligned}
 P4 &= [OHIP = "233GA", [OHIP \rightarrow PATID], 3] \\
 P5 &= [TestName = "whitebloodcount", [TestName \rightarrow Test], 3] \\
 P6 &= [Date = "Jan/04", [Date \rightarrow Dt], 4]
 \end{aligned}$$

Notice that, we can not simply translate the predicate terms independently without looking for the potential dependency from other terms. In the example, the term `fname = "Andrew"` does not have any correspondence assertion but by combining it with the term `lname = "Hall"`, we can translate the *subquery* (`lname = "Hall"`, `fname = "Andrew"`). Because the terms can be mapped with the correspondence assertion $lname, fname \rightarrow name$. After rewriting the selection formula we apply, the corresponding translation rule to each newly generated leaves of the *query condition tree*.

The query translation algorithm depends on mapping tables and correspondence assertions. The algorithm translates SPJ queries, where selection formula may contain both conjunctive and disjunctive normal form with negation. Sometimes a situation may arise where a query is not translatable. There are three reasons for this case. First, no mapping exists in the mapping table to translate a predicate term. Second, there is no correspondence assertion that maps a predicate term. Third, there is no rule to support the translation of a predicate atom. In all such cases the query is rejected.

6.1 Sound Translation of Queries

In peer database systems, the translation of a given query is not unique [14]. There could be many possible global queries for a particular query, because there is no certain control of query propagation in peer database networks. Also peers are free to join and leave the system at will. Therefore, in most cases, we do not get *complete answer* (meaning all the matching tuples in the network), but at least we get some answer (sound answer) which can be recognized as complete enough with respect to the set of active peers.

In this section we describe the soundness (correctness) of query translation. Soundness ensures that the translated query retrieves correct data from acquainted peers which are relevant to the result of the original query. Consider two peers P_1 and P_2 with export schemas $V_1[U_1] \subseteq DB_1[W_1]$ and $V_2 \subseteq DB_2[W_2]$, respectively. Assume that a query translation rule r and a mapping table m exist along with correspondence assertions between the peers. Suppose that a query q_1 is posed over V_1 and that q_2 is the translation of q_1 over V_2 . We use the notation $q_1 \rightarrow q_2$ to state that q_2 results from the translation of query q_1 . Intuitively, ensuring a correct translation means that the translation should be such that q_2 retrieves from P_2 only the data that are related to those that could be retrieved from query q_1 in peer P_1 . It is important to establish such a notion of correctness. Here, we extend the definition of correctness of query translation with respect to mapping tables given in [14].

Definition 3. [*Soundness w.r.t to Mapping Tables*] Let q_1, q_2 be queries over peer P_1 and P_2 , respectively; Let $q_1 = \sigma_E(R_1 \bowtie \dots \bowtie R_k)$, where E is a selection

formula and R_1, \dots, R_k are relations in P_1 . Then q_2 is a sound translation of q_1 with respect to a set $\mathcal{M} = \{m_1(X_1, Y_1), \dots, m_k(X_k, Y_k)\}$ of mapping tables, denoted by $q_1 \xrightarrow{\mathcal{M}} q_2$, where $X_i, i = 1..k$, and $att(q_1)$ are subsets of $\bigcup_{i=1}^k X_i$, if for every relation instance r_2 of P_2 and $t_2 \in q_2(r_2)$, there exists a valuation ρ of \mathcal{M} , and a tuple $t \in \sigma_E(\rho(\mathcal{M})), i = 1 \dots k$ such that $\pi_{att(q_2)}(t) = t_2$.

In this paper the translation algorithm incorporates the mapping tables into data association rules. Formally we must extend the definition of soundness to incorporate the translation rules seen in Section 5.

Definition 4 (Soundness w.r.t Translation Rules). Let q_1, q_2 be queries over peer P_1 and P_2 , respectively; Let $q_1 = \sigma_E(R_1 \bowtie \dots \bowtie R_k)$, where E is a selection formula and R_1, \dots, R_k are relations in P_1 . Then query q_2 is a sound translation of query q_1 with respect to a set of translation rules \mathcal{R} and correspondence assertions $CA = \{ca_1, \dots, ca_n\}$ between P_1 and P_2 , denoted by $q_1 \xrightarrow{\mathcal{R}, CA} q_2$, if $att(q_1) \subseteq att(CA)$, $att(q_2) \subseteq att(CA)$, and for every relation instance I_2 of P_2 and $t_2 \in q_2(I_2)$, there exists a tuple $t \in q_1(I_1)$, where I_1 is an instance of P_1 , such that for all $r \in \mathcal{R}$,

1. if r is a merge rule (See rule (5.2)), then, for all selection terms $\sigma_{B=y}$ mentioned in q_2 there is a complex selection term $\sigma_{\bigwedge A_i=x_i}$, $1 \leq i \leq n$, mentioned in q_1 , such that

$$y = \Pi_B(f_M(R(A_1, \dots, A_n), B) \bowtie T_{\bigwedge A_i=x_i}).$$

2. if r is a data association rule, then use Definition 3.

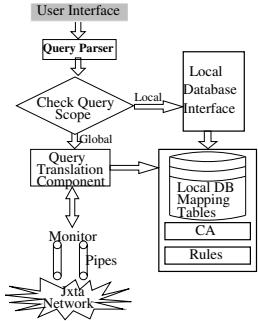
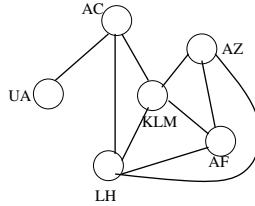
The cases of the separation and data conversion rules are treated similarly to the merge rule.

7 Implementation

We implemented the query translation algorithm described in this paper. The architecture of the query translation framework is depicted in Figure 7. The translation process starts when a user poses a query through the graphical user interface and selects the global execution of the query. If the query is local then the query is processed locally. The main component of the architecture is the Query Translation Component which is the implementation of the algorithm in Figure 5.

The monitor component looks for incoming queries in the network. It receives and forwards queries to the acquainted peers.

The prototype P2P setting is shown in Figure 8. We use MySQL as our DBMS for the Local DB. Data have been collected from United Airline, Air Canada, KLM, Lufthansa, Air France and Alitalia flight information. There are around 50 mapping tables to map flight numbers, destinations, etc between partner airlines. Each mapping table contains an average of 150 records. We choose JXTA for implementation platform because it provides all resources and functionalities

**Fig. 7.** Architecture**Fig. 8.** The P2P Network

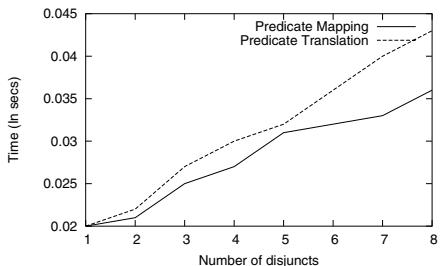
for developing P2P application. For example, it provides basic protocols and communication links (called pipes) between peers. It also provides basic peer functionalities such as creation a peer on a network; creation of messages and messages communication onto pipes, discover peers, creation of peer groups and join a peer group, etc. JXTA is an open network computing platform for P2P computing. It provides IP independent naming space to address peers and other resources. Java is used for the programming language. The P2P platform is simulated on IBM computers with windows XP operating system. The CPU is Pentium 4 3.0 GHz and RAM is 760MB.

7.1 Experimental Results

To evaluate our algorithm, we measure various run times. We first find mapping times of query predicates because query translation mainly depends on the size of predicate in the query. shows a query and a resulting translated query. Figure 10 shows the mapping times and predicate translation times of queries.

```

*****Original Query*****
select lname, fname, address from lh.customer where ((fno='LH482' OR fno='LH484')
OR (date='10/05/2003' AND depetime='02') AND dest='shanghai'
Parse Time=0.015
Elapsed time of Mapping:0.016
Elapsed time of Predicate Translation:0.062
Elapsed time of Query Translation:0.093
*****Translated Query:*****
select name, city, word from ac_passenger where (((fno='AC780') OR (fno='AC782')
OR (fno='AC784') OR (fno='AC785') OR (fno='AC712') OR (fno='AC724') OR (fno='
C716') OR (fno='AC218') OR (fno='AC220') OR (fno='AC224')) OR (((fno='AC780')
OR (fno='AC782') OR (fno='AC784') OR (fno='AC785') OR (fno='AC712') OR (fno='AC7
24') OR (fno='AC216') OR (fno='AC218') OR (fno='AC220') OR (fno='AC224')))) OR (((f
date='10/05/2003' AND depetime='02') AND dest='shanghai'))
  
```

**Fig. 9.** An output of a translated query**Fig. 10.** Predicate mapping and translation

We run this experiment with 8 queries where the number of search conditions gradually ranges from 1 to 8. That is, the first query has one condition,

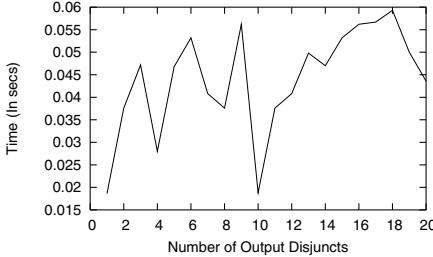


Fig. 11. Query translation time with size of output queries

```

wwwOriginal Query:
select fno from fno where fno='LH9766'
wwwTranslated Query:www
For Peer1:
select fno, date,deptime, dest, fms from ac where (((fno='AC70B') OR (fno='AC702') OR (fno='AC704') OR (fno='AC706') OR (fno='AC712') OR (fno='AC714') OR (fno='AC716') OR (fno='AC718') OR (fno='AC720') OR (fno='AC724')))
For Peer1m:
select fno, date,deptime, dest from kln where (((fno='KL061') OR (fno='KL0643') OR (fno='KL068') OR (fno='KL072') OR (fno='KL074') OR (fno='KL076') OR (fno='KL078') OR (fno='KL081') OR (fno='KL084') OR (fno='KL086') OR (fno='KL088') OR (fno='KL090') OR (fno='KL092') OR (fno='KL094') OR (fno='KL096') OR (fno='KL098') OR (fno='KL099')))
For Peer1z:
select fno, date,deptime, dest from az where (((fno='AZ0010') OR (fno='AZ0011') OR (fno='AZ0012') OR (fno='AZ0013') OR (fno='AZ0014') OR (fno='AZ0015') OR (fno='AZ0016') OR (fno='AZ0017') OR (fno='AZ0018') OR (fno='AZ0019') OR (fno='AZ0020') OR (fno='AZ0021') OR (fno='AZ0022') OR (fno='AZ0023') OR (fno='AZ0024') OR (fno='AZ0025') OR (fno='AZ0026') OR (fno='AZ0027') OR (fno='AZ0028') OR (fno='AZ0029') OR (fno='AZ0030') OR (fno='AZ0031') OR (fno='AZ0032') OR (fno='AZ0033') OR (fno='AZ0034') OR (fno='AZ0035') OR (fno='AZ0036') OR (fno='AZ0037') OR (fno='AZ0038') OR (fno='AZ0039') OR (fno='AZ0040') OR (fno='AZ0041') OR (fno='AZ0042') OR (fno='AZ0043') OR (fno='AZ0044') OR (fno='AZ0045') OR (fno='AZ0046') OR (fno='AZ0047') OR (fno='AZ0048') OR (fno='AZ0049') OR (fno='AZ0050') OR (fno='AZ0051') OR (fno='AZ0052') OR (fno='AZ0053') OR (fno='AZ0054') OR (fno='AZ0055') OR (fno='AZ0056') OR (fno='AZ0057') OR (fno='AZ0058') OR (fno='AZ0059') OR (fno='AZ0060') OR (fno='AZ0061') OR (fno='AZ0062') OR (fno='AZ0063') OR (fno='AZ0064') OR (fno='AZ0065') OR (fno='AZ0066') OR (fno='AZ0067') OR (fno='AZ0068') OR (fno='AZ0069') OR (fno='AZ0070') OR (fno='AZ0071') OR (fno='AZ0072') OR (fno='AZ0073') OR (fno='AZ0074') OR (fno='AZ0075') OR (fno='AZ0076') OR (fno='AZ0077') OR (fno='AZ0078') OR (fno='AZ0079') OR (fno='AZ0080') OR (fno='AZ0081') OR (fno='AZ0082') OR (fno='AZ0083') OR (fno='AZ0084') OR (fno='AZ0085') OR (fno='AZ0086') OR (fno='AZ0087') OR (fno='AZ0088') OR (fno='AZ0089') OR (fno='AZ0090') OR (fno='AZ0091') OR (fno='AZ0092') OR (fno='AZ0093') OR (fno='AZ0094') OR (fno='AZ0095') OR (fno='AZ0096') OR (fno='AZ0097') OR (fno='AZ0098') OR (fno='AZ0099')))
Line Elapsed:8.375

```

Fig. 12. Query translation for different peers

the second two condition, and so on. We see from the figure that the mapping times increase gradually. The queries are chosen in such a way that there are interdependency between predicate terms in the query. Nicks on the curves are points where there is a change in the amount of interdependency among predicate terms. between the time required to perform a query translation and the size, in terms of predicate terms, of the translated (or output) query. Figure 11 shows the translation times for queries which gradually produce a number of terms ranging from 1 to 20. The interesting point to notice from Figure 11 is that it is not the number of terms in the output query which a relevant factor for the running times to generate these terms. These times mainly depend on the number of terms in the input query and on the dependency between predicate terms.

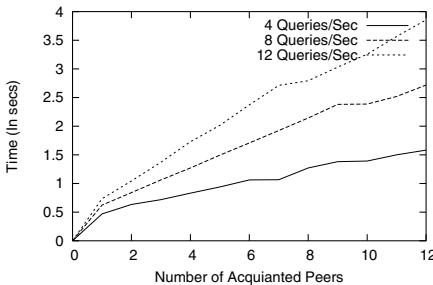


Fig. 13. Query translation time with number of peers

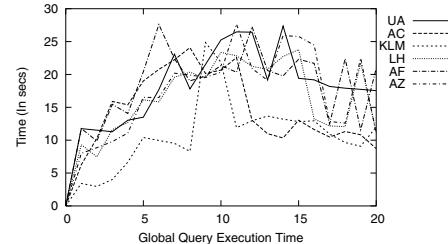


Fig. 14. Global query execution time

We also investigate the algorithm performance for query translation with the number of acquaintances. We experimented with 12 peers. We investigated the times required to translate one query (in a peer) for all the acquainted peers. Figure 12 gives a screenshot showing such a translation. We investigate with different input query frequencies. We notice that the translation times increase gradually with the number of peers and number of input queries per second. The Figure 13 shows the result.

We also investigate the execution of global queries generated from different peers in our P2P settings. We flooded the network generating 4 queries/sec from each peer with total 20 queries per peer. The total number of queries in the network was 624. The result is shown in Figure 14. The figure shows that the global queries of the KLM and LH peers finish first because they have more acquaintances than other peers in the settings. The global queries of the UA peer take the highest times because this peer is linked to the P2P network over one single acquaintance (with peer AC).

8 Related Work

Ooi et al. [16] present a peer-to-peer (P2P) distributed data sharing system called PeerDB. Query processing in PeerDB is performed through keyword matching and ranking using agents. The keyword-matching strategy in PeerDB may give irrelevant query reformulations because a keyword of a relation may match syntactically with keywords of attributes or relations of other peers without being a semantical match. The user must decide which queries are to be executed. In PeerDB, continuous user involvements are required before the user fetches required data from peers. In our approach, on the contrary, once acquaintances are in place, the user need not worry about them at query time.

The paper [9] introduces a data model called Local Relational Model (LRM) designed for P2P data management systems to describe relationships between two peer databases. The acquaintances between two peers are based upon the definition of coordination formulas and domain relations within the system. The main goals of the data model to support semantic interoperability in the absence of global schema.

The Piazza system [17] provides a solution for query answering in a peer-to-peer environment, where the associations between peers are expressed as either global-as-view (GAV) or local-as-view (LAV) mapping. All these are schema (as opposed to our instance) level mappings.

An approach for data coordination avoiding the assumptions of global schema is introduced in [21]. The authors of [21] introduce the notion of group and define it as a set of nodes, which are able to answer queries about a certain topic. Each group has a node called Group Manager (GM), which is in charge of the management of the metadata in order to run the group [21]. According to their proposal, each query must pass through the group manager. The paper does not mention how to choose a node as a group manager.

9 Conclusion

In this paper we investigated a data sharing strategy through query translation based on syntactic and instance level mappings. We addressed some translation rules based on these mappings. Our strategy can translate a query if there is appropriate correspondence assertions between the schema elements of acquainted peers. A future work, we plan to investigate the approach as a query service

built on top of a large scale peer data management system. We also plan to investigate the dynamic inference of new correspondence assertions from existing ones. Such a dynamic inference mechanism seems necessary to avoid doing so manually when peers join/leave the system.

References

1. The JXTA Project. <http://www.jxta.org>
2. M. Boyd, S. Kittivoravikitkul, C. Lazaridis, P. McBrien, N. Rizopoulos. AutoMed: A BAV Data Integration System for Heterogeneous Data Sources. In *CAiSE*, 2004
3. R. Domenig, K.R. Dittrich. Query Explorativeness for Integrated Search in Heterogeneous Data Sources. In *CAiSE*, 2002
4. L. Serafini, F. Giunchiglia, J. Molopoulos, and P. Bernstein. Local Relational Model:a logocal formalization of database coordination. Technical Report, Informatica e Telecomunicazioni, University of Trento, 2003.
5. M. Lenzerini. Data Integration: A Theoretical Prespective. In *PODS*, 2001.
6. R. J. Miller, L. M. Haas and M. Hernndez. Schema Mapping as Query Discovery. In *VLDB*, 2000.
7. Z. Ives A. Halevy, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? in *webdb*. In *WebDB*, 2001.
8. M. Arenas, V. Kantere, A. Kementsietsidis, and I. Kiringa. The hyperion project: From data integration to data coordination. In *ACM SIGMOD RECORD*, 2003.
9. P. Bernstein, F. Giunchiglia, A. Kementsietsidis, and J. Mylopoulos. Data management for peer-to-peer computing: A vision. In *WebDB*, 2002.
10. C.-C. K. Chang and H. Garcia-Molina. Mind your vocabulary: Query mapping across heterogeneous information source. In *SIGMOD*, 1999.
11. F. Giunchiglia and I. Zaihrayeu. Making peer databases interact-a vision. In *CIA*, 2002.
12. A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management system. In *ICDE*, 2003.
13. V. Kantere, I. Kiringa, and J. Mylopoulos. Coordinating peer databases using ECA rules. In *P2P DBIS*, 2003.
14. A. Kementsietsidis and M. Arenas. Data sharing through query translation in autonomous systems. In *VLDB*, 2004.
15. A. Kementsietsidis, M. Arenas, and R.J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *SIGMOD*, 2003.
16. W. S. Ng, B. C. Ooi, K. L. Tan, and A. Y. Zhou. PeerDB:A p2p-based system for distributed data sharing. In *Data Engineering*, 2003.
17. I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, and X. Dong. The piazza peer data management project. In *ICDE*, 2003.
18. P. reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Middleware*, 2003.
19. E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. The coDB Robust Peer-to-Peer Database System. In *SEDB*, pages 382-393, 2004.
20. E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. Queries and Updates in the coDB Peer to Peer Database System. In *VLDB*, pages 1277-1280, 2004.
21. F. Giunchiglia and I. Zaihrayeu. Making Peer Databases Interact-A vision for an Architecture Supporting Data Coordination. In *CIA*, pages 18-35, 2002.

On the Usage of Global Document Occurrences in Peer-to-Peer Information Systems

Odysseas Papapetrou, Sebastian Michel, Matthias Bender,
and Gerhard Weikum

Max-Planck Institut für Informatik, 66123 Saarbrücken, Germany
`{odysseas, smichel, mbender, weikum}@mpi-inf.mpg.de`

Abstract. There exist a number of approaches for query processing in Peer-to-Peer information systems that efficiently retrieve relevant information from distributed peers. However, very few of them take into consideration the overlap between peers: as the most popular resources (e.g., documents or files) are often present at most of the peers, a large fraction of the documents eventually received by the query initiator are duplicates. We develop a technique based on the notion of *global document occurrences* (GDO) that, when processing a query, penalizes frequent documents increasingly as more and more peers contribute their local results. We argue that the additional effort to create and maintain the GDO information is reasonably low, as the necessary information can be piggybacked onto the existing communication. Early experiments indicate that our approach significantly decreases the number of peers that have to be involved in a query to reach a certain level of recall and, thus, decreases user-perceived latency and the wastage of network resources.

1 Introduction

1.1 Motivation

The peer-to-peer (P2P) approach, which has become popular in the context of file-sharing systems such as Gnutella or KaZaA, allows handling huge amounts of data in a distributed and self-organizing way. In such a system, all peers are equal and all of the functionality is shared among all peers so that there is no single point of failure and the load is evenly balanced across a large number of peers. These characteristics offer enormous potential benefits for search capabilities powerful in terms of scalability, efficiency, and resilience to failures and dynamics. Additionally, such a search engine can potentially benefit from the intellectual input (e.g., bookmarks, query logs, etc.) of a large user community.

One of the key difficulties, however, is to efficiently select promising peers for a particular information need. While there exist a number of strategies to tackle this problem, most of them ignore the fact that popular documents are typically present at a reasonable fraction of peers. In fact, experiments show that often promising peers are selected because they share the same high-quality documents. Consider a query for all songs by a famous artist like Madonna. If, as in

many of today's systems, every selected peer contributes its best matches only, you will most likely end up with many duplicates of popular and recent songs, when instead you would have been interested in a bigger variety of songs. The same scenario holds true in an information retrieval context where returning only the k best matches for a query is even more common. Popular documents then are uselessly contributed as query results by each selected peer, wasting precious local resources and disqualifying other relevant documents that eventually might not be returned at all. The size of the combined result eventually presented to the query initiator (after eliminating those duplicates), thus, is unnecessarily small.

1.2 Contribution

We propose a technique based on the notion of *global document occurrences* (GDO) that, when processing a query, penalizes frequent documents increasingly as more and more peers contribute their local results. The same approach can also be used prior to the query execution when selecting promising peers for a query. We discuss the additional effort to create and maintain the GDO information and present early experiments indicating that our approach significantly decreases the number of peers that have to be involved in a query to reach a certain level of recall. Thus, taking overlap into account when performing query routing is a great step towards the feasibility of distributed P2P search.

Section 2 gives an overview of related research in the different fields that we touch with our work. Section 3 gives a short introduction on Information Retrieval basics necessary for the remainder of this paper. Section 4 presents the architecture of MINERVA, our distributed P2P search engine that was used for our experiments. Section 5 introduces the notion of GDO and discusses its application at several stages of the querying process. Section 6 illustrates a number of experiments to show the potential of our approach. Section 7 concludes and briefly discusses future research directions.

2 Related Work

Recent research on P2P systems, such as Chord [1], CAN [2], Pastry [3], P2P-Net [4], or P-Grid [5] is based on various forms of distributed hash tables (DHTs) and supports mappings from keys, e.g., titles or authors, to locations in a decentralized manner such that routing scales well with the number of peers in the system. Typically, in a network of n nodes, an exact-match key lookup can be routed to the proper peer(s) in at most $O(\log n)$ hops, and no peer needs to maintain more than $O(\log n)$ routing information. These architectures can also cope well with failures and the high dynamics of a P2P system as peers join or leave the system at a high rate and in an unpredictable manner. However, the approaches are limited to exact-match, single keyword queries on keys. This is insufficient when queries should return a ranked result list of the most relevant approximate matches [6].

In recent years, many approaches have been proposed for collection selection in distributed IR, among the most prominent the decision-theoretic framework by [7], the GLOSS method presented in [8], and approaches based on statistical language models [9,10]. [11] gives an overview of algorithms for distributed IR style result merging and database content discovery. [7] presents a formal decision model for database selection in networked IR. [12] investigates different quality measures for database selection. [13,14] study scalability issues for a distributed term index. None of the presented techniques incorporates overlap detection into the selection process.

[15] describes a permutation-based technique for efficiently estimating set similarities for informed content delivery. [16] proposes a hash-based synopsis data structure and algorithms to support low-error and high-confident estimates for general set expressions. Bloom [17] describes a data structure for succinctly representing a set in order to support membership queries. [18] proposes compressed Bloom filters that improve performance in a distributed environment where network bandwidth is an issue.

[19] describes the use of statistics in ranking data sources with respect to a query. They use probabilistic measures to model overlap and coverage of the mediated data sources, but do not mention how to acquire these statistics. In contrast, we assume these statistics being generated by the participating peers (based on their local collections) and present a DHT based infrastructure to make these statistics globally available.

[20] considers novelty and redundancy detection in a centralized, document-stream based information filtering system. Although the technique presented seems to be applicable in a distributed environment for filtering the documents at the querying peer, it is not obvious where to get these documents from. In a large-scale system, it seems impossible to query all peers and to process the documents.

[21,22] have also worked on overlap statistics in the context of collection selection. They present a technique to estimate coverage and overlap statistics by query classification and data mining and use a probing technique to extract features from the collections. Expecting that data mining techniques will be very heavy for the envisioned, highly-dynamic application environment, we adopt a different philosophy.

In a prior work [23] we propose a Bloom filter based technique to estimate the mutual collection overlap. While in this earlier work, we use Bloom filters to estimate the mutual overlap between peers, we now use the number of global document occurrences of the documents in a collection to estimate the contribution of this collection to a particular query. These approaches can be seen as orthogonal and can eventually be combined to form even more powerful systems.

3 Information Retrieval Basics

Information Retrieval (IR) systems keep large amounts of unstructured or weakly structured data, such as text documents or HTML pages, and offer search

functionalities for delivering documents relevant to a query. Typical examples of IR systems include web search engines or digital libraries; in the recent past, relational database systems are integrating IR functionality as well.

The search functionality is typically accomplished by introducing measures of similarity between the query and the documents. For text-based IR with keyword queries, the similarity function typically takes into account the number of occurrences and relative positions of each query term in a document. Section 3.1 explains the concept of *inverted index lists* that support an efficient query execution and section 3.2 introduces one of the most popular similarity measures, the so-called *TF*IDF* measure. For further reading, we refer the reader to [6,24].

3.1 Inverted Index Lists

The concept of inverted index lists has been developed in order to efficiently identify those documents in the dataset that contain a specific query term. For this purpose, all terms that appear in the collection form a tree-like index structure (often a B^+ -tree or a trie) where the leafes contain a list of unique document identifiers for all documents that contain this term (Figure 1). Conceptually, these lists are combined by intersection or union for all query terms to find candidate documents for a specific query. Depending on the exact query execution strategy, the lists of document identifiers may be ordered according to the document identifiers or according to a score value to allow efficient pruning.

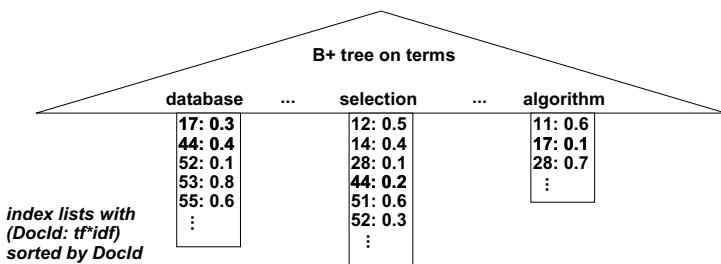


Fig. 1. B^+ Tree of Inverted Index Lists

3.2 *TF * IDF* Measure

The number of occurrences of a term t in a document d is called *term frequency* and typically denoted as $tf_{t,d}$. Intuitively, the significance of a document increases with the number of occurrences of a query term. The number of documents in a collection that contain a term t is called *document frequency* (df_t); the *inverse document frequency* (idf_t) is defined as the inverse of df_t . Intuitively, the relative importance of a query term decreases as the number of documents that contain this term increases, i.e., the term offers less differentiation between the documents. In practice, these two measures may be normalized (e.g., to values

between 0 and 1) and damped using logarithms. A typical representative of this family of $tf * idf$ formulae that calculates the weight $w_{i,f}$ of the i -th term in the j -th document is

$$w_{i,j} := \frac{tf_{i,j}}{\max_t\{tf_{t,j}\}} * \log\left(\frac{N}{df_i}\right)$$

where N is the total number of documents in the collection.

In recent years, other relevance measures based on statistical language models and probabilistic IR have received wide attention [7,25]. For simplicity and because our focus is on P2P distributed search, we use the still most popular $tf * idf$ scoring family in this paper.

4 MINERVA

We briefly introduce MINERVA¹, a fully operational distributed search engine that we have implemented and that serves as a valuable testbed for our work[26,27]. We assume a P2P collaboration in which every peer is autonomous and has a local index that can be built from the peer's own crawls or imported from external sources and tailored to the user's thematic interest profile. The index contains inverted lists with URLs for Web pages that contain specific keywords.

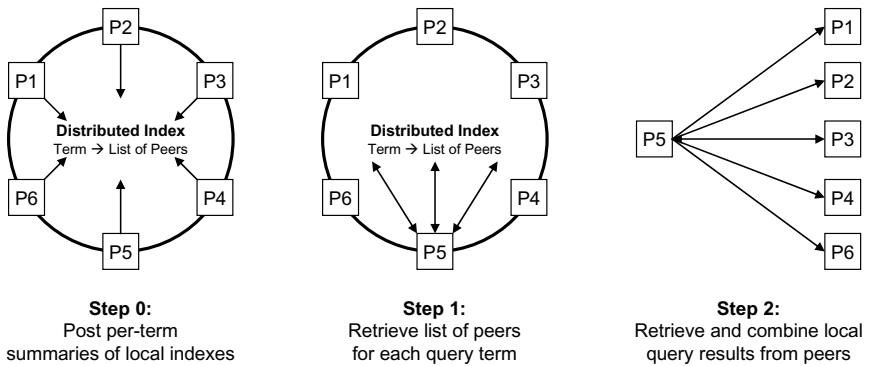


Fig. 2. MINERVA System Architecture

A conceptually global but physically distributed directory, which is layered on top of a Chord-style Dynamic Hash Table (DHT), holds compact, aggregated information about the peers' local indexes and only to the extent that the individual peers are willing to disclose. We only use the most basic DHT functionality, $lookup(key)$, that returns the peer currently responsible for key . Doing so, we partition the term space, such that every peer is responsible for

¹ Project homepage available at <http://www.minerva-project.org>

a randomized subset of terms within the global directory. For failure resilience and availability, the entry for a term may be replicated across multiple peers.

Directory maintenance, query routing, and query processing work as follows (cf. Figure 2). In a preliminary step (step 0), every peer publishes a summary (*Post*) about every term in its local index to the directory. A hash function is applied to the term in order to determine the peer currently responsible for this term. This peer maintains a *PeerList* of all postings for this term from peers across the network. Posts contain contact information about the peer who posted this summary together with statistics to calculate IR-style measures for a term (e.g., the size of the inverted list for the term, the maximum average score among the term’s inverted list entries, or some other statistical measure). These statistics are used to support the query routing process, i.e., determining the most promising peers for a particular query.

The querying process for a multi-term query proceeds as follows: a query is executed locally using the peer’s local index. If the result is considered unsatisfactory by the user, the querying peer retrieves a list of potentially useful peers by issuing a *PeerList request* for each query term to the underlying overlay-network directory (step 1). Using database selection methods from distributed IR and metasearch [11], a number of promising peers for the complete query is computed from these PeerLists. This step is referred to as *query routing*. Subsequently, the query is forwarded to these peers and executed based on their local indexes (*query execution*; step 2). Note that this communication is done in a pairwise point-to-point manner between the peers, allowing for efficient communication and limiting the load on the global directory. Finally, the results from the various peers are combined at the querying peer into a single result list.

The goal of finding high-quality search results with respect to precision and recall cannot be easily reconciled with the design goal of unlimited scalability, as the best information retrieval techniques for query execution rely on large amounts of document metadata. Posting only compact, aggregated information about local indexes and using appropriate query routing methods to limit the number of peers involved in a query keeps the size of the global directory manageable and reduces network traffic, while at the same time allowing the query execution itself to rely on comprehensive local index data. We expect this approach to scale very well as more and more peers jointly maintain the moderately growing global directory.

The approach can easily be extended in a way that multiple distributed directories are created to store information beyond local index summaries, such as information about local bookmarks, information about relevance assessments (e.g., derived from peer-specific query logs or click streams), or explicit user feedback. This information could be leveraged when executing a query to further enhance result quality.

4.1 Query Routing

Database selection has been a research topic for many years, e.g. in distributed IR and metasearch [11]. Typically, the expected result quality of a collection is

estimated using precomputed statistics, and the collections are ranked accordingly. Most of these approaches, however, are not directly applicable in a true P2P environment, as

- the number of peers in the system is substantially higher (10^x peers as opposed to 10-20 databases)
- the system evolves dynamically, i.e. peers enter or leave the system autonomously at their own discretion at a potentially high rate
- the results from remote peers should not only be of high quality, but also complementary to the results previously obtained from one's local search engine or other remote peers

In [26,28], we have adopted a number of popular existing approaches to fit the requirements of such an environment and conducted extensive experiments in order to evaluate the performance of these naive approaches.

As a second step, we have extended these strategies using estimators of mutual overlap among collections [23] using bloom filters [17]. Preliminary experiments show that such a combination can outperform popular approaches based on quality estimation only, such as CORI [11].

We also want to incorporate the fact that every peer has its own local index, e.g., by using implicit-feedback techniques for automated query expansion (e.g., using the well-known IR technique of pseudo relevance feedback [29] or other techniques based on query logs [30] and click streams [31]). For this purpose, we can benefit from the fact that each peer executes the query locally first, and also the fact that each peer represents an actual user with personal preferences and interests. For example, we want to incorporate local user bookmarks into our query routing [28], as bookmarks represent strong recommendations for specific documents. Queries could be exclusively forwarded to thematically related peers with similarly interested users, to improve the chances of finding *subjectively* relevant pages.

Ultimatively, we want to introduce a sophisticated *benefit/cost* ratio when selecting remote peers for query forwarding. For the benefit estimation, it is intuitive to consider such measures as described in this section. Defining a meaningful cost measure, however, is an even more challenging issue. While there are techniques for observing and inferring network bandwidth or other infrastructural information, expected response times (depending on the current system load) are changing over time. One approach is to create a distributed Quality-of-Service directory that, for example, holds moving averages of recent peer response times.

4.2 Query Execution

Query execution based on local index lists has been an intensive field of research for many years in information retrieval. A good algorithm should avoid reading inverted index lists completely, but limit the effort to $O(k)$ where k is the number of desired results. In the IR and multimedia-search literature, various algorithms have been proposed to accomplish this. The best known general-purpose method

for top- k queries is Fagin's threshold algorithm (TA) [32], which has been independently proposed also by Nepal et al. [33] and Güntzer et al. [34]. It uses index lists that are sorted in descending order of term scores under the additional assumption that the final score for a document is calculated using a monotone aggregation function (such as a simple sum function). TA traverses all inverted index lists in a round-robin manner, i.e., lists are mainly traversed using sorted accesses. For every new document d encountered, TA uses random accesses to calculate the final score for d and keeps this information in document candidate set. Since TA additionally keeps track of a higher bound for documents not yet encountered, the algorithm terminates as soon as this bound assures that no unseen document can enter the candidate set. Probabilistic methods have been studied in [35] that can further improve the efficiency of index processing.

As our focus is on the distributed aspect of query processing, we will not focus on query execution in this paper. Our approaches to be introduced in the upcoming sections are orthogonal to this issue and can be applied to virtually any query execution strategy.

5 Global Document Occurrences (GDO)

We define the *global document occurrence* of a document d (GDO_d) as the number of peers that contain d , i.e., as the number of occurrences of d within the network. This is substantially different from the notion of *global document frequency* of a term t (which is the number of documents that contain t) and from the notion of *collection frequency* (which is typically defined as the number of collections that contain documents that contain t).

The intuition behind using GDO when processing a query is the fact that GDO can be used to efficiently estimate the probability that a peer contains a certain document and, thus, the probability that a document is contained in at least one of a set of peers. Please note the obvious similarity to the *TF * IDF* measure, that weights the relative importance of a query term t using the number of documents that contain t as an estimation of the popularity of t , favoring rare terms over popular (and, thus, less distinctive and discriminative) terms. Similarly, the GDO approach weights the relative popularity of a document within the union of all collections. If a document is highly popular (i.e., occurs in most of the peers), it is considered less important both when selecting promising peers (query routing) and when locally executing the query (query execution). In contrast, rare documents receive a higher relative importance.

5.1 Mathematical Reasoning

The proposed approach will get clearer if we describe the reasoning behind it. Suppose that we are running a single-keyword query, and that each document d in our collection has a precomputed relevance to a term t (noted as $DocumentScore(d, t)$). When searching for the top- k documents, a P2P system would ask some of its peers for documents, which determine the relevant documents locally, and merge the results.

This independent document selection has the disadvantage that it does not consider overlapping results. For example, one relevant document might be so common, that every peer returns it as result. This reduces the recall for a query, as the document is redundant for all but the first peer. In fact, massive document replication is common in real P2P systems, so duplicate results frequently occur. This effect can be described with a mathematical model, which can be used to improve document retrieval.

Assuming a uniform distribution of documents among the peers, the probability that a given peer has a certain document d can be estimated by

$$P_H(d) = \frac{GDO(d)}{\#\text{peers}}.$$

Now consider a sequence of peers $\langle p_1, \dots, p_\lambda \rangle$. The probability that a given document d held by p_λ is fresh, i.e. not already occurs in one of the previous peers, can be estimated by

$$P_F^\lambda(d) = (1 - P_H(d))^{\lambda-1}.$$

This probability can now be used to re-evaluate the relevance of documents: If it is likely that a previously queried peer has already returned a document, the document is no longer relevant. Note that we introduce a slight inaccuracy here: We only used the probability that one of the previously asked peers has a document, not the probability that it has also returned the document. Thus we would be interested in the probability that a document has not been returned before $P_{NR}^\lambda(d)$. However the error introduced is reasonably small: for all documents $P_{NR}^\lambda(d) \geq P_F^\lambda(d)$. For the relevant documents $P_{NR}^\lambda(d) \approx P_F^\lambda(d)$, as the relevant documents will be returned by the peers. Therefore we only underestimate (and, thus, punish) the probability for irrelevant documents, which is not too bad, as they were irrelevant anyway.

Now this probability can be used to adjust the scores according to the GDO. The most direct usage would be to discard a document d during retrieval with a probability of $(1 - P_F^\lambda(d))$, but this would produce non-deterministic behavior. Instead we adjust the DocumentScores of a document d with regard to a term t by aggregating the scores and the probability; for simplicity, we multiply them in our current experiments.

$$\text{DocumentScore}'(d, t) = \text{DocumentScore}(d, t) * P_F^\lambda(d)$$

This formula reduces the scores for frequent documents, which avoids duplicate results. Note that $P_F^\lambda(\text{document})$ decreases with λ , thus frequent documents are still returned by peers asked early, but discarded by the following peers.

5.2 Apply GDO to Query Routing

In most of the existing approaches to query routing, the quality of a peer is estimated using per-term statistics about the documents that are contained in its

collection. Popular approaches include counting the number of documents that contain this term (*document frequency*), or summing up the document scores for all these documents (*score mass*). These term-specific scores are combined to form an aggregated *PeerScore* with regard to a specific query. The peers are ordered according to their PeerScore to form a peer *ranking* that determines an order in which the peers will be queried.

The key insight of our approach to tackle the problem of retrieving duplicate documents seems obvious: the probability of a certain document being contained in at least one of the involved peers increases with the number of involved peers. Additionally, the more popular the document, the higher the probability that it is contained in one of the first peers to contribute to a query. Thus, the impact of such documents to the PeerScore should decrease as the number of involved peers increases.

If a candidate peer in the ranking contains a large fraction of popular documents, it would be increasingly unwise to query this peer at later stages of the ranking, as the peer might not have any fresh (i.e., previously unseen) documents to offer. In contrast, if no peers have been queried yet, then a peer should not be punished for containing popular documents, as we certainly *do* want to retrieve those documents. We suggest an extension that is applicable to almost all popular query routing strategies and calculates the PeerScore of a peer depending on its position in the peer ranking.

For this purpose, we modify the score of each document in a collection with different biases, one for each position in a peer ranking². In other words, there is no longer only *one* DocumentScore for each document, but rather several DocumentScores corresponding to the potential ranks in a peer ranking. Remember from the previous section, that the DocumentScore of a document d with regard to term t is calculated using the following formula:

$$\text{DocumentScore}'(d, t, \lambda) = \text{DocumentScore}(d, t) * P_F^\lambda(d)$$

where λ is the position in the peer ranking (i.e., the number of peers that have already contributed to the query before), and $P_F^\lambda(d)$ is the probability that this document is not contained in any of the previously contributing collections.

From this set of DocumentScores, each peer now calculates *separate* term-specific scores (i.e., the scores that serve as subscores when calculating PeerScores in the process of Query Routing) corresponding to the different positions in a peer ranking by combining the respectively biased document scores. In the simplest case where the PeerScore was previously calculated by summing up the scores for all relevant documents, this means that now one of these sums is calculated for every rank λ :

$$\text{score}(p, t, \lambda) = \sum_{d \in D_p} \text{DocumentScore}'(d, t, \lambda)$$

² Please note that, for techniques that simply count the number of documents, all scores are initially set to 1.

where D_p denotes the document collection of p . Instead of including only one score in each term-specific post, now a *list* of the term-specific peer scores $score(p, t, \lambda)$ is included in the statistics that is published to the distributed directory. Figure 3 shows some extended statistics for a particular term. The numbers shown in the boxes left to the scores represent the respective ranks in a peer ranking. Please note that the term-specific score of a peer decreases as the document scores for its popular documents decrease with the ranking position. Prior experiments have shown that typically involving only 2-3 peers in a query already yields a reasonable recall; we only calculate $score(p, t, \lambda)$ for $\lambda \leq 10$ [26] as we consider asking more than 10 peers very rare and not compatible with our goal of system scalability. The calculation itself of this magnitude of DocumentScores is negligible.

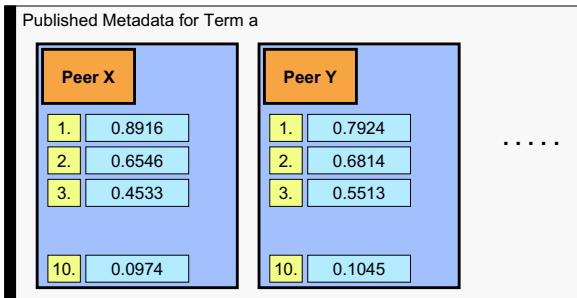


Fig. 3. Extended Term-specific scores for different ranking positions

Please also note that this process does not require the selected peers to locally execute the queries sequentially, but it allows for the parallel query execution of all peers involved: after identifying the desired number of peers and their ranks in the peer ranking, the query initiator can contact all other peers simultaneously and include their respective ranks in the communication. Thus, the modification of the standard approach using GDOs does not cause additional latencies or bandwidth consumption.

The additional network resource consumption needed for our proposed approach is relatively small if conducted in a clever manner. Instead of distributing the GDO counters across the peers using random hashing on unique document identifiers, we propose to maintain the counters at peers that are responsible for a representative term within the document, (e.g., the first term or the most frequent term). Doing so, we can easily piggyback the GDO-related communication when publishing the Posts and, in turn, can immediately receive the current GDO values for the same documents. The GDO values are then cached locally and used to update the local DocumentScores, that will eventually be used when publishing our Posts again. The Posts itself become slightly larger as more than one score value is now included in a Post; this will typically fit within the existing network message avoiding extra communication.

5.3 Apply GDO to Query Execution

The peers that have been selected during query routing can additionally use GDO-dependent biases to penalize popular documents during their local query execution. The later a peer is involved in the processing of a query, the higher punishing impact this GDO-dependent bias should have as popular documents are likely to be considered at prior peers. For this purpose, each peer re-weights the DocumentScores obtained by its local query execution with the GDO-values for the documents.

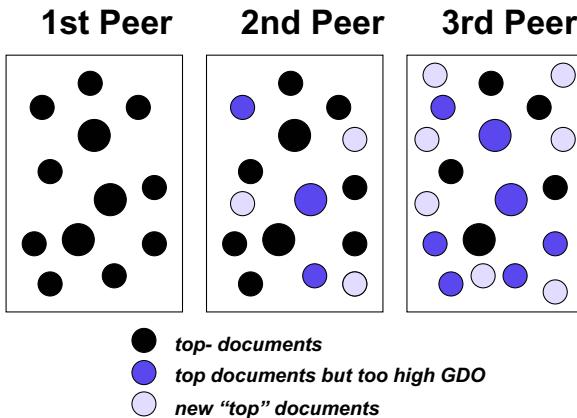


Fig. 4. The impact of GDO-enhanced query execution

Figure 4 shows the impact of the GDO-based local query execution³.

The additional cost caused by our approach within the query execution step is negligible. As the GDO values are cached locally as described in a previous section, the DocumentScores can easily be adjusted on-line using a small number of basic arithmetic operations.

5.4 Building and Maintaining GDO

All the approaches introduced above build on top of a directory that globally counts the number of occurrences of each document. When a new peer joins the network, it updates GDO for all its documents (i.e., increment the respective counters) and retrieves the GDO values for the computation of its biased scores at low extra cost.

We propose the usage of the existing distributed DHT-based directory to maintain the GDO values in a scalable way. In a naive approach, the document space is partitioned across all peers using globally unique document identifiers, e.g., by applying a hash function to their URLs and maintaining the counter at

³ In case you see a 79 in the right figure, please contact your local ophthalmologist immediately.

the DHT peer that is responsible for this identifier (analogously to the term-specific statistics that are maintained independently in parallel). This naive approach would require two messages for each document per peer (one when the peer enters and one when the peer leaves the network), which results to $O(n)$ messages for the whole system, where n is the number of document instances.

However, the advanced approach of piggybacking this information onto existing messages almost avoids additional messages completely. In fact, when a peer enters the network, no additional messages are required for the GDO maintenance, as all messages are piggybacked in the process of publishing Post objects to the directory.

To cope with the dynamics of a Peer-to-Peer system, in which peers join and leave the system autonomously and without prior notice, we propose the following technique. Each object in the global directory is assigned a TTL (time-to-live) value, after which it is discarded by the maintaining peer. In turn, each peer is required to re-send its information periodically. This fits perfectly with our local caching of GDO values, as these values can be used when updating the Post objects. This update process, in turn, again updates the local GDO values.

6 Experiments

6.1 Benchmarks

We have generated two synthetic benchmarks. The first benchmark includes 50 peers and 1000 unique documents, while the second benchmark consists of 100 peers and 1000 unique documents. We assign term-specific scores to the documents following a Zipf[36] distribution (skewness $\alpha = 0.8$), as in real world we often find documents that were highly relevant with regard to one term, but practically irrelevant (with a very low score) with regard to the remaining terms. The assumption that the document scores follow Zipf's law is widely accepted in information retrieval literature.

The *document replication* follows a Zipf distribution, too. This means that most documents are assigned to a very small number of peers (i.e., have a low GDO value) and only very few documents are assigned to a large number of peers (i.e., have a high GDO value). Please note that, although the GDOs and the document scores of the documents were following a Zipf distribution, the two distributions were not connected. This means that we do not expect a document with a very high importance for one term to be also highly replicated. We do not believe that this would create real-world document collections as we know from personal experiences that the most popular documents are not necessarily the most relevant documents.

6.2 Evaluated Strategies

In our experimental evaluation, we compare six different strategies. All strategies consist of the query routing part and the query execution part. For query routing, our baseline algorithm for calculating the PeerScore of a peer p works as follows:

- $score(p, t) = \sum_{d \in D_p} DocumentScore(d, t)$, i.e., the (unbiased) score mass of all relevant documents in p 's collection D_p
- $PeerScore(p, q) = \sum_{t \in q} score(p, t)$, i.e., the sum over all term-specific scores for all terms t contained in the query q

For the query execution part, the synthetically created DocumentScores were derived by summing up the (synthetically assigned) term-specific scores described above. At both stages, query routing and query processing, we can either choose a standard (non-GDO) approach or our GDO-enhanced approach, yielding a total of four strategies. The GDO values were provided to each strategy using global knowledge of our data.

In addition, we employ two other strategies that use a mod- k sampling-based query execution technique to return fresh documents: In the query execution process, the peers will return only documents with $(DocumentId \bmod \kappa) = \lambda$ where κ is the total number of peers that are going to be queried (i.e. top-10), and λ is the number of peers that have already been queried.

6.3 Evaluation Methodology

We run several three-term queries using the six strategies introduced above. In each case, we send the query to the top-10 peers suggested by each approach, and collect the local top-20 documents from each peer. Additionally, we run the queries on a combined collection of all peers to retrieve the global top-100 documents that serves as a baseline for our strategies.

We use four metrics to assess the quality of each strategy:

- the number of *distinct* retrieved documents, i.e., after eliminating duplicates
- the score mass of all distinct retrieved document⁴
- the number of distinct retrieved top-100 documents
- the score mass of distinct retrieved top-100 documents

6.4 Results

The experiments are conducted on both benchmark collections. Due to space limitations, we only present the results for the 50-peer setup; the results of the 100-peer setup are very similar.

The GDO-enhanced strategies show significant performance gains. Figure 5 shows the number of distinct retrieved documents, while Figure 6 shows the aggregated score masses for these documents. Figure 7 shows the number of distinct retrieved top-100 documents; Figure 8 shows the corresponding score masses. While all other strategies outperform the baseline strategy, it is interesting to notice that query execution can obviously draw more benefit from the GDO-enhancement than query routing can; if applied to query routing only, our GDO-approach does not show significant performance improvements. This

⁴ Note that, by design, the same document is assigned the same score at different peers.

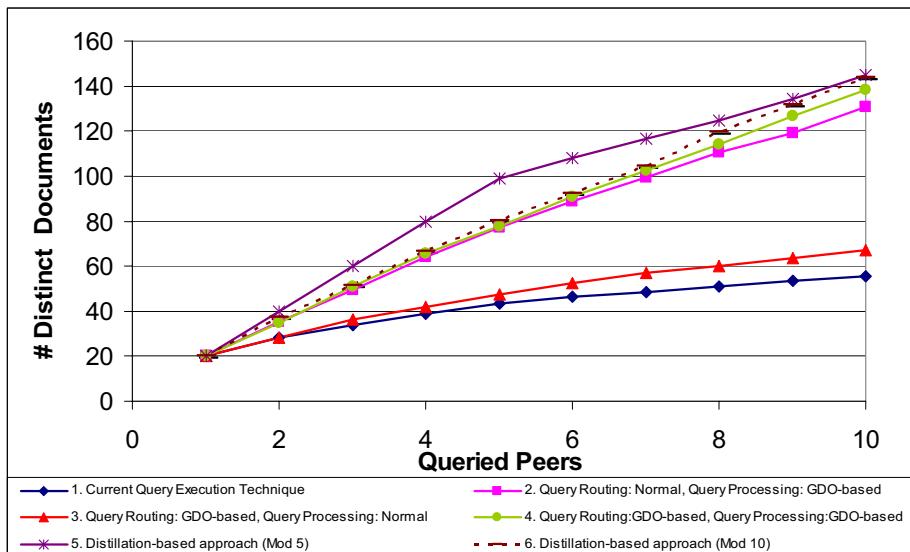


Fig. 5. Distinct documents retrieved with regard to the number of queried peers

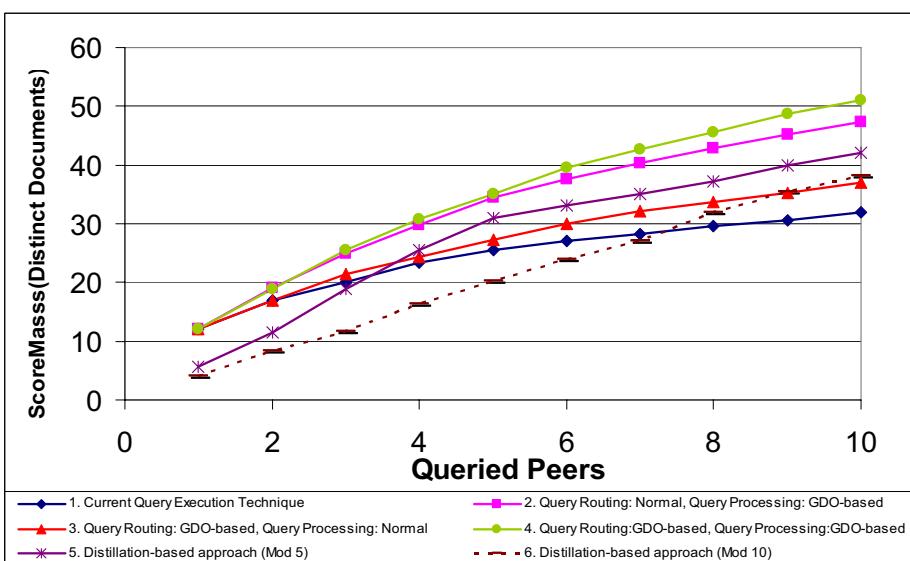


Fig. 6. Score mass of the retrieved documents with regard to the number of queried peers

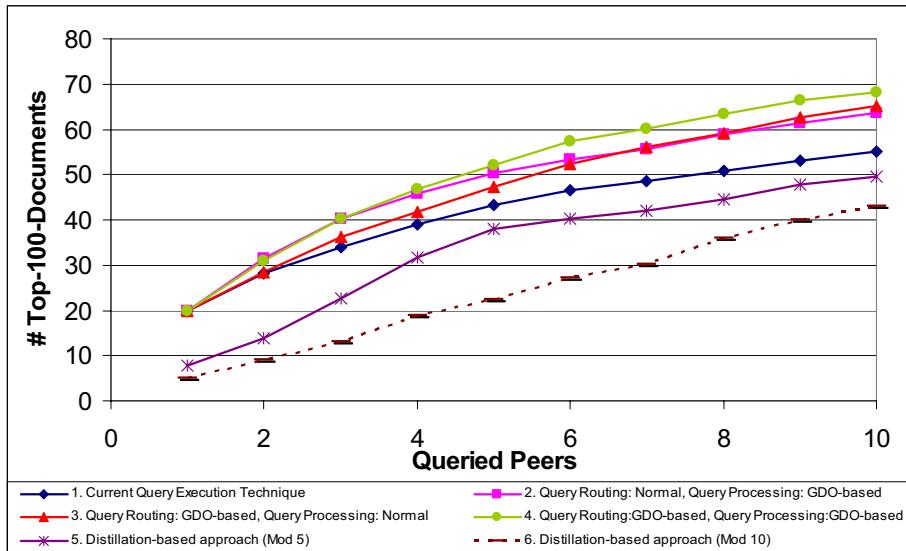


Fig. 7. Distinct documents from global top-100 with regard to the number of queried peers

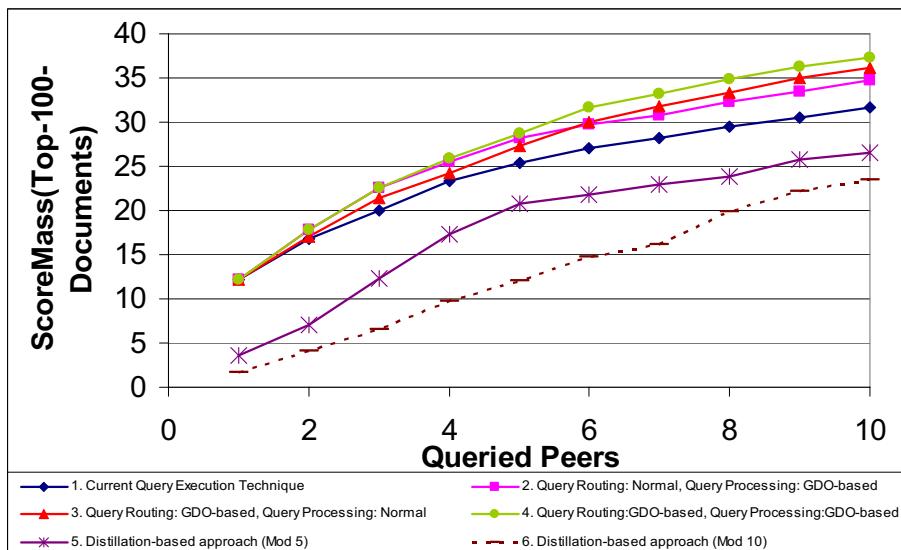


Fig. 8. Score mass of distinct retrieved documents from global top-100 with regard to the number of queried peers

does not come as a surprise and is partly due to the nature of our benchmark. For larger peer populations showing significant mutual overlap, we expect the GDO-enhanced query routing to outperform the baseline strategy in a more impressive way. On the other hand, the query execution technique has a great impact on the number of distinct documents. Using GDO-enhancement, popular documents are discarded from the local query results, giving place to other (otherwise not considered) documents.

The naive mod- κ approaches are quite successful in retrieving distinct documents; however, they perform bad if we evaluate the quality of the returned documents by calculating score masses. On the other hand, using the two-way GDO-enhanced strategy (both GDO-routing and GDO-query processing) combines many fresh documents with high scores for our query, resulting in a significant recall improvement.

7 Conclusion and Future Work

This work presents an approach toward improving the query processing in Peer-to-Peer Information Systems. The approach is based on the notion of Global Document Occurrences (GDO) and aims at increasing the number of uniquely retrieved high-quality documents without imposing significant additional network load or latency. Our approach can be applied both at the stage of query routing (i.e., when selecting promising peers for a particular query) and when locally executing the query at these selected peers. The addition cost caused to build and maintain the required statistical information is small and our approach is expected to scale very well with a growing network. Early experiments show the potential of our approach, significantly increasing the recall experienced in our settings.

We are currently working on experiments on real data obtained from focused web crawls, which exactly fits our environment of peers being users with individual interest profiles. Also, a more thorough study of the resource consumption of our approach is under way. One central point of interest is the directory maintenance cost; in this context, we evaluate strategies that do not rely on periodically resending all information, but on explicit GDO increment/decrement messages. Using a time-sliding window approach, this might allow us to even more efficiently estimate the GDO values.

References

1. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the ACM SIGCOMM 2001, ACM Press (2001) 149–160
2. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: Proceedings of ACM SIGCOMM 2001, ACM Press (2001) 161–172

3. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). (2001) 329–350
4. Buchmann, E., Böhm, K.: How to Run Experiments with Large Peer-to-Peer Data Structures. In: Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, USA. (2004)
5. Aberer, K., Punceva, M., Hauswirth, M., Schmidt, R.: Improving data access in p2p systems. *IEEE Internet Computing* **6** (2002) 58–67
6. Chakrabarti, S.: Mining the Web: Discovering Knowledge from Hypertext Data. Morgan Kaufmann, San Francisco (2002)
7. Fuhr, N.: A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems* **17** (1999) 229–249
8. Gravano, L., Garcia-Molina, H., Tomasic, A.: Gloss: text-source discovery over the internet. *ACM Trans. Database Syst.* **24** (1999) 229–264
9. Si, L., Jin, R., Callan, J., Ogilvie, P.: A language modeling framework for resource selection and results merging. In: Proceedings of CIKM02, ACM Press (2002) 391–397
10. Xu, J., Croft, W.B.: Cluster-based language models for distributed retrieval. In: Research and Development in Information Retrieval. (1999) 254–261
11. Callan, J.: Distributed information retrieval. Advances in information retrieval, Kluwer Academic Publishers. (2000) 127–150
12. Nottelmann, H., Fuhr, N.: Evaluating different methods of estimating retrieval quality for resource selection. In: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, ACM Press (2003) 290–297
13. Grabs, T., Böhm, K., Schek, H.J.: Powerdb-ir: information retrieval on top of a database cluster. In: Proceedings of CIKM01, ACM Press (2001) 411–418
14. Melnik, S., Raghavan, S., Yang, B., Garcia-Molina, H.: Building a distributed full-text index for the web. *ACM Trans. Inf. Syst.* **19** (2001) 217–241
15. Byers, J., Considine, J., Mitzenmacher, M., Rost, S.: (Informed content delivery across adaptive overlay networks. In Proceedings of ACM SIGCOMM, 2002.)
16. Ganguly, S., Garofalakis, M., Rastogi, R.: Processing set expressions over continuous update streams. In: SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, ACM Press (2003) 265–276
17. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* **13** (1970) 422–426
18. Mitzenmacher, M.: Compressed bloom filters. *IEEE/ACM Trans. Netw.* **10** (2002) 604–612
19. Florescu, D., Koller, D., Levy, A.Y.: Using probabilistic information in data integration. In: The VLDB Journal. (1997) 216–225
20. Zhang, Y., Callan, J., Minka, T.: Novelty and redundancy detection in adaptive filtering. In: SIGIR '02: Proceedings of the 25th annual international ACM SI-GIR conference on Research and development in information retrieval, ACM Press (2002) 81–88
21. Nie, Z., Kambhampati, S., Hernandez, T.: Bibfinder/statminer: Effectively mining and using coverage and overlap statistics in data integration. In: VLDB. (2003) 1097–1100
22. Hernandez, T., Kambhampati, S.: (Improving text collection selection with coverage and overlap statistics. pc-recommended poster. WWW 2005. Full version available at <http://rakaposhi.eas.asu.edu/thomas-www05-long.pdf>)

23. Bender, M., Michel, S., Triantafillou, P., Weikum, G., Zimmer, C.: Improving collection selection with overlap awareness in p2p systems. In: Proceedings of the SIGIR Conference. (2005)
24. Manning, C.D., Schütze, H.: Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge, Massachusetts (1999)
25. Croft, W.B., Lafferty, J.: Language Modeling for Information Retrieval. Volume 13. Kluwer International Series on Information Retrieval (2003)
26. Bender, M., Michel, S., Weikum, G., Zimmer, C.: The MINERVA project: Database selection in the context of P2P search. (In: BTW 2005)
27. Bender, M., Michel, S., Weikum, G., Zimmer, C.: Minerva: Collaborative p2p search. In: Proceedings of the VLDB Conference (Demonstration). (2005)
28. Bender, M., Michel, S., Weikum, G., Zimmer, C.: Bookmark-driven query routing in peer-to-peer web search. In Callan, J., Fuhr, N., Nejdl, W., eds.: Proceedings of the SIGIR Workshop on Peer-to-Peer Information Retrieval. (2004) 46–57
29. Buckley, C., Salton, G., Allan, J.: The effect of adding relevance information in a relevance feedback environment. In: SIGIR, Springer-Verlag (1994)
30. Luxenburger, J., Weikum, G.: Query-log based authority analysis for web information search. In: WISE04. (2004)
31. Srivastava et al., J.: Web usage mining: Discovery and applications of usage patterns from web data. SIGKDD Explorations **1** (2000) 12–23
32. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Symposium on Principles of Database Systems. (2001)
33. Nepal, S., Ramakrishna, M.V.: Query processing issues in image (multimedia) databases. In: ICDE. (1999) 22–29
34. Guntzer, U., Balke, W.T., Kiesling, W.: Optimizing multi-feature queries for image databases. In: The VLDB Journal. (2000) 419–428
35. Theobald, M., Weikum, G., Schenkel, R.: Top-k query evaluation with probabilistic guarantees. VLDB (2004)
36. Zipf, G.K.: Human behavior and the principle of least effort. addison-wesley press. (1949)

An Approach for Clustering Semantically Heterogeneous XML Schemas

Pasquale De Meo¹, Giovanni Quattrone¹,
Giorgio Terracina², and Domenico Ursino¹

¹ DIMET, Università “Mediterranea” di Reggio Calabria, Via Graziella,
Località Feo di Vito, 89060 Reggio Calabria, Italy

² Dipartimento di Matematica, Università della Calabria, Via Pietro Bucci,
87036 Rende (CS), Italy
`{demeo, quattrone, ursino}@unirc.it,`
`terracina@mat.unical.it`

Abstract. In this paper we illustrate an approach for clustering semantically heterogeneous XML Schemas. The proposed approach is driven mainly by the semantics of the involved Schemas that is defined by means of the interschema properties existing among concepts represented therein. An important feature of our approach consists of its capability to be integrated with almost all the clustering algorithms already proposed in the literature.

1 Introduction

Clustering is the process of grouping a set of physical or abstract objects into classes of similar objects called *clusters* [11], in such a way that those objects belonging to the same cluster are as similar as possible, whereas those ones belonging to different clusters are as dissimilar as possible.

Clustering has its roots in many areas, including Data Mining, Statistics, Biology, and Machine Learning. Its applications are extremely various and range from Economy to Finance, from Biology to Sociology, and so on. Clustering can play a key role also in the Web; in fact, in this scenario, numerous applications that largely benefit of clustering, ranging from the definition of user classes to the construction of multi-thematic portals, have been proposed [3].

In the Web context, a specific activity in which clustering can play a key role consists of grouping semantically heterogeneous information sources. In fact, currently, the number of information sources available on the Web is exponentially increasing, along with their semantic heterogeneity. As a consequence, it appears extremely important the definition of approaches for clustering them into homogeneous classes.

On the contrary, as for data representation format and data exchange, the World Wide Web Consortium foretells, for the future, a certain uniformity and, to this purpose, proposes the usage of the XML language.

The growing importance of both clustering and XML stimulated, in the past, various researchers to study clustering techniques for XML sources [6,14,16,19].

However, many of these techniques were “structural”, i.e., they aimed at defining groups of structurally similar XML sources [6,16]. Clearly, the structural similarity of two information sources is an indicator of their semantic similarity; however, often, it could be not sufficient [4,10,17].

In the literature various methods for defining the information sources semantics have been proposed; one of the most commonly adopted methods consists of the exploitation of the so-called *interschema properties*, [2,4,17,21], i.e., terminological and structural relationships existing among concepts belonging to different information sources. The most important interschema properties are synonymies, homonymies, hyponymies/hypernymies, overlappings, and subschema similarities.

In the literature also some approaches for clustering XML sources, taking into account their semantic similarities, have been proposed [14,19]. However, in these approaches, source similarity is determined by considering only concept similarities (that, in the context of interschema properties, correspond to synonymies). In our opinion these approaches are extremely interesting; however, they could be further refined if, in addition to synonymies, also other interschema properties, such as hyponymies and overlappings, would be considered.

This paper aims at providing a contribution in this setting; in fact, it presents an approach for clustering semantically heterogeneous information sources; the proposed approach takes into account not only synonymies but also hyponymies and overlappings.

We point out that the present paper has not been conceived for defining a new clustering algorithm; on the contrary, it aims at allowing the application of most of the existing algorithms to our reference context. As a matter of fact, in the literature, a great number of clustering algorithms, characterized by extremely various features already exists, and other ones will be presumably proposed in the future. As a consequence, allowing the application of all these algorithms to our reference context would provide the user with the availability of a large variety of clustering tools, characterized by different peculiarities.

The key for reaching such a result is the exploitation of the so called *dissimilarity matrix* [11]; this is, in fact, the data structure which almost all the clustering algorithms already proposed in the literature operate on. The rows and the columns of this matrix represent the objects to cluster; its generic element $M[i, j]$ denotes the distance, i.e., the dissimilarity, between the objects i and j . Generally, $M[i, j]$ is a non-negative number that is as closer to 0 as i and j are similar.

The proposed approach exploits interschema properties for finding the dissimilarity degree between two XML Schemas and, consequently, for constructing the dissimilarity matrix. Since some clustering algorithms require the involved objects to be represented as points in a metric space (see Section 2), in order to allow the applicability of the maximum possible number of clustering algorithms to our reference context, we define the dissimilarity among XML Schemas by exploiting a suitable euclidean space.

2 Description of the Proposed Approach

2.1 Introduction

As pointed out in the Introduction, the main focus of the proposed approach is the clustering of semantically heterogeneous XML Schemas.

Our approach receives: (*i*) a set $SchemaSet = \{S_1, S_2, \dots, S_n\}$ of XML Schemas to cluster; (*ii*) a dictionary IPD storing the interschema properties (synonymies, hyponymies/hypernyms, and overlappings) involving concepts belonging to Schemas of $SchemaSet$. These properties can be computed by exploiting any approach proposed in the literature for this purpose; specifically, in our experiments, we have adopted the approach described in [7], for the computation of synonymies, and that described in [8], for the computation of hyponymies and overlappings. These two approaches are not only structural but also semantic; the semantics of a concept is determined by considering its neighborhoods, i.e., the other concepts related to it and the corresponding kinds of relationships. For instance, two concepts are considered synonymous if their neighborhoods are similar. Both the two approaches require the support of a thesaurus, e.g., WordNet.

In the following we shall assume that IPD is ordered on the basis of the names of the involved elements and attributes; if this is not the case, our approach preliminarily applies a suitable sorting algorithm on it.

Before providing a detailed description of the behaviour of our approach, it is necessary to introduce some definitions that will be largely exploited in the following.

Let S_i be an XML Schema. An x -component of S_i is an element or an attribute of S_i ; it is characterized by its name, its typology (stating if it is a simple element, a complex element or an attribute), and its data type. The set of x -components of S_i is called $XCompSet(S_i)$. In the following we shall denote with $P = \sum_{i=1}^n |XCompSet(S_i)|$ the total number of x -components belonging to the Schemas of $SchemaSet$.

We define now some functions that will be extremely useful in the following; they receive two x -components x_ν and x_μ and return a boolean value; these functions are:

- $identical(x_\nu, x_\mu)$, that returns *true* if and only if x_ν and x_μ are two synonymous x -components having the same name, the same typology, and the same data type;
- $strong(x_\nu, x_\mu)$, that returns *true* if and only if x_ν and x_μ are two synonymous x -components and, at the same time, $identical(x_\nu, x_\mu) = false$;
- $weak(x_\nu, x_\mu)$, that returns *true* if and only if x_ν and x_μ are two x -components related by either an hyponymy/hypernymy or an overlapping property.

Proposition 1. Let $SchemaSet = \{S_1, S_2, \dots, S_n\}$ be a set of XML Schemas; let P be the total number of x -components relative to the Schemas of $SchemaSet$; finally, let x_ν and x_μ be two x -components belonging to two distinct Schemas of

SchemaSet. The computation of $identical(x_\nu, x_\mu)$, $strong(x_\nu, x_\mu)$, and $weak(x_\nu, x_\mu)$ costs $O(\log P)$.

Proof. At most one kind of interschema properties can exist between two x-components of different Schemas. As a consequence, the maximum cardinality of IPD is $O(P^2)$. The computation of each function mentioned above implies the search of the corresponding pair in IPD . Since this dictionary is ordered, it is possible to apply the binary search. This costs $O(\log(P^2)) = O(2 \log P) = O(\log P)$. \square

Starting from the functions defined previously, it is possible to construct the following support dictionaries:

- *Identity Dictionary ID*, defined as:

$$ID = \{\langle x_\nu, x_\mu \rangle \mid x_\nu, x_\mu \in \bigcup_{i=1}^n XCompSet(S_i), identical(x_\nu, x_\mu) = true\};$$

- *Strong Similarity Dictionary SSD*, defined as:

$$SSD = \{\langle x_\nu, x_\mu \rangle \mid x_\nu, x_\mu \in \bigcup_{i=1}^n XCompSet(S_i), strong(x_\nu, x_\mu) = true\};$$

- *Weak Similarity Dictionary WSD*, defined as:

$$WSD = \{\langle x_\nu, x_\mu \rangle \mid x_\nu, x_\mu \in \bigcup_{i=1}^n XCompSet(S_i), weak(x_\nu, x_\mu) = true\}.$$

The construction of these dictionaries is carried out in such a way that they are always ordered w.r.t. the names of the involved x-components.

Proposition 2. Let $SchemaSet = \{S_1, S_2, \dots, S_n\}$ be a set of XML Schemas; let P be the total number of x-components relative to Schemas of $SchemaSet$. The construction of ID , SSD , and WSD costs $O(P^2 \times \log P)$.

Proof. The construction of each dictionary is carried out by verifying the corresponding function for each of the $O(P^2)$ pairs of x-components. Proposition 1 states that this verification costs $O(\log P)$; as a consequence, the total cost of the construction of each dictionary is $O(P^2 \times \log P)$. \square

2.2 Construction of the Dissimilarity Matrix

As specified in the Introduction, the main focus of our approach is the construction of the dissimilarity matrix. In fact, once this structure has been constructed, it is possible to apply on it a large variety of clustering algorithms already proposed in the literature. In order to allow the application of the maximum possible number of clustering algorithms, we have decided to exploit a metrics for measuring the dissimilarity between two XML Schemas.

Since involved XML Schemas could be semantically heterogeneous and since we want to group them on the basis of their relative semantics, our definition of metrics must necessarily be very different from the classical ones; specifically, in our case, it must be strictly dependent on the interschema properties that are the way we exploit for defining the inter-source semantics.

In order to introduce our notion of metrics we have exploited a suitable, multi-dimensional euclidean space. It has P dimensions, one for each x-component of the involved XML Schemas; in the following it will be denoted by the symbol \Re^P .

An XML Schema S_i can be represented in \Re^P by means of a point $Q_i \equiv [q_1^i, q_2^i, \dots, q_\nu^i, \dots, q_P^i]$. The value of the generic coordinate q_ν^i is obtained by means of the following formula:

$$q_\nu^i = \xi(x_\nu) \times \psi(x_\nu, S_i, ID, SSD, WSD)$$

$\xi(x_\nu)$ discriminates the complex elements w.r.t. the simple ones and the attributes. This is necessary because a complex element is much more characterizing than either a simple element or an attribute in defining the semantics of a concept. ξ is defined in the following way:

$$\xi(x_\nu) = \begin{cases} 1 & \text{if } x_\nu \text{ is a complex element} \\ \gamma & \text{if } x_\nu \text{ is either a simple element or an attribute} \end{cases}$$

Here, γ belongs to the real interval $[0, 1]$.

$\psi(x_\nu, S_i, ID, SSD, WSD)$ indicates how much S_i is capable of representing the semantics expressed by the concept associated with x_ν . Clearly, its capability is maximum if it contains x_ν or an x-component identical to x_ν ; its capability decreases if there exists only a synonym of x_ν in it; its capability further decreases if it contains an x-component related to x_ν only by either an hyponymy/hypernymy or an overlapping property. ψ is defined in the following way:

$$\psi(x_\nu, S_i, ID, SSD, WSD) = \begin{cases} 1 & \text{if } x_\nu \in XCompSet(S_i) \text{ or} \\ & \quad \text{if } \exists x_\mu \in XCompSet(S_i) \mid \langle x_\nu, x_\mu \rangle \in ID \\ \alpha & \text{if } \exists x_\mu \in XCompSet(S_i) \mid \langle x_\nu, x_\mu \rangle \in SSD \\ \beta & \text{if } \exists x_\mu \in XCompSet(S_i) \mid \langle x_\nu, x_\mu \rangle \in WSD \\ 0 & \text{otherwise} \end{cases}$$

Here, α and β belong to the real interval $[0..1]$; moreover, we have that, generally, $\beta < \alpha < 1$. An experimental study on the values of α , β and γ is illustrated in Section 3.3.

Proposition 3. Let $SchemaSet = \{S_1, S_2, \dots, S_n\}$ be a set of XML Schemas; let P be the total number of x-components relative to the Schemas of $SchemaSet$. Let S_i be a Schema of $SchemaSet$. The worst case time complexity for determining the point Q_i associated with S_i in \Re^P is $O(|XCompSet(S_i)| \times P \times \log P)$.

Proof. In order to determine Q_i the functions ξ and ψ must be evaluated for each dimension of \Re^P . Let us consider the ν^{th} dimension. The cost of ξ is constant. ψ requires to perform a search in ID , SSD and WSD for each x-component of S_i . Since these dictionaries are ordered, this search can be performed in $O(\log P)$. As a consequence, the total cost of the function ψ is $O(|XCompSet(S_i)| \times \log P)$.

This evaluation must be repeated for each of the P dimensions; as a consequence, the total cost for determining Q_i is $O(|XCompSet(S_i)| \times P \times \log P)$. \square

Corollary 1. Let $SchemaSet = \{S_1, S_2, \dots, S_n\}$ be a set of XML Schemas; let P be the total number of x-components relative to the Schemas of $SchemaSet$. The worst case time complexity for determining in \Re^P the points associated with all Schemas of $SchemaSet$ is $O((\sum_{i=1}^n |XCompSet(S_i)|) \times P \times \log P) = O(P^2 \times \log P)$. \square

We are now able to introduce our notion of distance between two XML Schemas and, consequently, to construct the dissimilarity matrix. Specifically, the distance between two XML Schemas S_i and S_j , belonging to $SchemaSet$, is computed by determining the euclidean distance between the corresponding points in \Re^P and by normalizing this distance in such a way to obtain a value in the real interval $[0..1]$, as required by the clustering algorithms. This is obtained by means of the following formula:

$$d(Q_i, Q_j) = \frac{\sqrt{\sum_{\nu=1}^P (q_{\nu}^i - q_{\nu}^j)^2}}{\sqrt{\sum_{\nu=1}^P (\xi(x_{\nu}))^2}}$$

Here, the numerator represents the classical euclidean distance between two points; the denominator represents the maximum distance possibly existing between two points in the considered space and it is exploited for performing the normalization¹.

Proposition 4. Let $SchemaSet = \{S_1, S_2, \dots, S_n\}$ be a set of XML Schemas; let P be the total number of x-components relative to the Schemas of $SchemaSet$. The worst case time complexity for constructing the dissimilarity matrix is $O(n^2 \times P)$.

Proof. Let $QSet = \{Q_1, Q_2, \dots, Q_n\}$ be the set of points in \Re^P associated with S_1, \dots, S_n ; in order to construct the dissimilarity matrix, it is necessary to compute the distance for each of the $O(n^2)$ pairs of points in $QSet$.

The computation of this distance requires the computation of the sums at the numerator (whose cost is $O(P)$) and at the denominator (whose cost is, again, $O(P)$) of $d(Q_i, Q_j)$.

Therefore, the total cost of the matrix construction is $O(n^2) \times O(2P) = O(n^2 \times P)$. \square

The previous reasonings show that, for constructing the dissimilarity matrix, two phases are necessary. The former aims at determining the points in the metric space corresponding to the involved XML Schemas; this activity costs $O(P^2 \times \log P)$ (see Corollary 1). The latter aims at computing the distances among the previously defined points and costs $O(n^2 \times P)$ (see Proposition 4).

¹ Recall that $q_{\nu}^i = \xi(x_{\nu}) \times \psi(x_{\nu}, S_i, ID, SSD, WSD)$ and that the maximum (resp., minimum) value of ψ is equal to 1 (resp., 0).

2.3 Application of the Pre-existing Clustering Algorithms on the Constructed Dissimilarity Matrix

Once the dissimilarity matrix has been defined, it is possible to apply on it a large variety of clustering algorithms previously proposed in the literature. These differ for their time complexity, for their result accuracy, as well as for their behaviour. Therefore, it is clear that the choice of the clustering algorithm to be adopted in a certain domain strictly depends on its main features. In order to evaluate this fact we have applied three clustering algorithms, characterized by different features, on our dissimilarity matrix. For implementation purposes, we have chosen to apply three algorithms available in WEKA [23], one of the most popular Data Mining tools; specifically, we have chosen to apply K-Means, Expectation Maximization, and Farthest First Traversal,.

In this section we provide a brief overview of the behaviour of these algorithms when applied to our reference context.

K-Means [15]. When applied to our reference context, K-Means receives a parameter k and partitions the set of points of \Re^P in k clusters.

The worst case time complexity of K-Means is $O(n \times k \times t)$, where n is the cardinality of *SchemaSet*, k is the number of clusters, and t is the number of iterations necessary for the algorithm to converge. Typically, k and t are much smaller than n ; therefore, the worst case time complexity of K-Means can be considered linear against the cardinality of *SchemaSet*; for this reason K-Means is relatively scalable in clustering large sets of XML Schemas.

A difficulty in the application of K-Means to our context regards its sensitivity to noise and outliers: this implies that, if there exist some Schemas semantically very different from the others, K-Means could return not particularly satisfactory results. Another drawback of K-Means consists of its necessity to preventively know the best value for k ; if this information is not available, a try-and-check approach should be exploited for determining it. Clearly, this would increase the time necessary to the algorithm for providing the final results.

Expectation Maximization [9,24]. Expectation Maximization (hereafter, *EM*) models involved objects as a collection of k Gaussians², where k is the number of clusters to be derived. For each involved object, *EM* computes a degree of membership to each cluster.

The implementation of *EM* is very similar to that of K-Means. As with K-Means, *EM* begins with an initial guess of the cluster centers (*Expectation* step), and iteratively refines them (*Maximization* step). It terminates when a parameter, measuring the quality of obtained clusters, no longer shows significant increases. *EM* is guaranteed to converge to a local maximum, that often coincides with the global one.

An important feature of this algorithm is its capability of modelling quite a rich set of cluster shapes. Moreover, it can be instructed to determine by itself

² Although the Gaussian models are those generally used, other distributions are possible.

the best number of clusters to be derived, even if the user can directly specify such an information, if he wants.

The quite refined statistical model underlying *EM* allows it to often obtain optimal results; for this reason *EM* is frequently adopted in a large variety of application contexts. Moreover, its capability of automatically determining the best number of clusters makes it particularly suited for our reference context.

Farthest First Traversal [13]. The basic idea of this algorithm is to get k points out of n , that are mutually “far” from each other; in our reference context, the points to cluster are the points of \Re^P associated with XML Schemas.

The algorithm operates as follows: first it randomly selects a point Q_1 and puts it into the so-called “Traversed Set” TS . After this, it performs $k - 1$ iterations for constructing TS ; during each iteration it inserts into TS the point Q_i having the maximum distance from TS ; the distance of Q_i from TS is defined as $d(Q_i, TS) = \min_{Q_j \in TS} d(Q_i, Q_j)$, where $d(Q_i, Q_j)$ could be any dissimilarity measure between two points (in our approach it is the dissimilarity measure between two points associated with two XML Schemas defined in Section 2.2).

After TS has been constructed, each point $Q_i \in TS$ is chosen as the centroid of a cluster; then, for each point $Q_k \notin TS$, the algorithm computes its distance from the various centroids and puts Q_k in the cluster whose centroid has the minimum distance from it.

Farthest First Traversal requires in input the number k of clusters to be constructed; moreover, the quality of its results might be influenced by the choice of the initial point Q_1 of TS . However, the worst case time complexity of this algorithm is $O(n \times k)$, where n is the cardinality of *SchemaSet* and k is the number of clusters to be obtained. As a consequence, it is scalable and particularly suited in application contexts, like ours, where objects to be clustered could be very numerous.

3 Experiments

3.1 Description of the Exploited Information Sources

In order to verify the validity of our approach we have performed various experiments. Specifically, we have considered 97 XML Schemas belonging to various application contexts, such as Biomedical Data, Project Management, Property Register, Industrial Companies, Universities, Airlines, Scientific Publications, and Biological Data.

These Schemas have been derived from specific Web sites or public sources. As an example, some XML Schemas relative to Biomedical Data have been derived from <http://www.biomediator.org>. Some of the Schemas relative to Project Management, Property Register, and Industrial Companies have been derived from Italian Central Government Office information sources and are shown at the address <http://www.mat.unical.it/terracina/tests.html>. Some of the Schemas relative to Universities have been downloaded from the address <http://anghai.cs.uiuc.edu/archive/domains/courses.html>. Schemas relative to Airlines have been

Table 1. Main features of the XML Schemas adopted in our experiments

<i>Application context</i>	<i>Number of Schemas</i>	<i>Maximum depth of Schemas</i>	<i>Minimum, Average and Maximum Number of x-components</i>	<i>Minimum, Average and Maximum Number of complex elements</i>
Biomedical Data	33	9	12 - 25 - 44	3 - 9 - 18
Project Management	9	6	35 - 40 - 46	5 - 6 - 9
Property Register	6	6	61 - 72 - 77	13 - 15 - 17
Industrial Companies	15	5	20 - 26 - 48	5 - 7 - 10
Universities	15	7	12 - 16 - 20	3 - 5 - 9
Airlines	2	4	12 - 13 - 13	4 - 4 - 4
Scientific Publications	2	6	17 - 18 - 18	8 - 9 - 9
Biological Data	15	9	230 - 322 - 658	33 - 55 - 221

found in [18]. Schemas relative to Scientific Publications have been supplied by the authors of [14]. Finally, Schemas relative to Biological Data have been downloaded from specialized sites; among them we cite <http://smi-web.stanford.edu/projects/helix/pubs/ismb02/schemas/>. The main features of the XML Schemas we have adopted in our experiments are described in Table 1.

3.2 Description of the Adopted Measures

The accuracy measures of a clustering approach can be subdivided into: (*i*) *external measures*, that compare the results obtained by the clustering approach into examination with the clusters defined by a domain expert and considered correct; (*ii*) *internal measures*, that evaluate the capability of the tested approach to produce homogeneous clusters.

External measures. In the following we introduce (and tailor to our reference context) some of the most popular external measures for the evaluation of clustering approaches [1,22].

Let $SchemaSet$ be the set of XML Schemas which the clustering task must be performed on; we indicate with $ClSet^* = \{Cl_1^*, Cl_2^*, \dots, Cl_l^*\}$ the set of correct classes defined by a domain expert, and with $ClSet = \{Cl_1, Cl_2, \dots, Cl_k\}$ the set of clusters produced by the algorithm to evaluate. The accuracy measures we have considered are:

- *Precision* (hereafter, Pre). The Precision of a cluster Cl_j w.r.t. a class Cl_i^* is defined as $Pre(Cl_i^*, Cl_j) = \frac{|Cl_j \cap Cl_i^*|}{|Cl_j|}$. The total Precision of a clustering approach, when applied on $SchemaSet$, is defined as:

$$Pre = \sum_{i=1}^{|ClSet^*|} \left[\frac{|Cl_i^*|}{|SchemaSet|} \times (\max_{1 \leq j \leq |ClSet|} Pre(Cl_i^*, Cl_j)) \right]$$

- *Recall* (hereafter, Rec). The Recall of a cluster Cl_j w.r.t. a class Cl_i^* is defined as $Rec(Cl_i^*, Cl_j) = \frac{|Cl_j \cap Cl_i^*|}{|Cl_i^*|}$. The total Recall of a clustering approach, when applied on $SchemaSet$, is defined as:

$$Rec = \sum_{i=1}^{|ClSet^*|} \left[\frac{|Cl_i^*|}{|SchemaSet|} \times (\max_{1 \leq j \leq |ClSet|} Rec(Cl_i^*, Cl_j)) \right]$$

- *F-Measure*. F-Measure represents the harmonic mean between Precision and Recall; it is defined as $F\text{-Measure}(Cl_i^*, Cl_j) = 2 \cdot \frac{Pre(Cl_i^*, Cl_j) \cdot Rec(Cl_i^*, Cl_j)}{Pre(Cl_i^*, Cl_j) + Rec(Cl_i^*, Cl_j)}$. The total F-Measure of a clustering approach, when applied on $SchemaSet$, is defined as:

$$F\text{-Measure} = \sum_{i=1}^{|ClSet^*|} \left[\frac{|Cl_i^*|}{|SchemaSet|} \times \left(\max_{1 \leq j \leq |ClSet|} F\text{-Measure}(Cl_i^*, Cl_j) \right) \right]$$

- *Overall*. Overall measures the post-match effort needed for adding false negatives and removing false positives from the set of similarities returned by the system to evaluate. It is defined as: $Overall(Cl_i^*, Cl_j) = Rec(Cl_i^*, Cl_j) \times \left(2 - \frac{1}{Pre(Cl_i^*, Cl_j)} \right)$. The total Overall of a clustering approach, when applied on *SchemaSet*, is defined as:

$$Overall = \sum_{i=1}^{|ClSet^*|} \left[\frac{|Cl_i^*|}{|SchemaSet|} \times \left(\max_{1 \leq j \leq |ClSet|} Overall(Cl_i^*, Cl_j) \right) \right]$$

- *Entropy*. Entropy provides a measure of the purity of clusters w.r.t. classes; it is defined as:

$$Entropy = \sum_{j=1}^{|ClSet|} \left[\frac{|Cl_j|}{|SchemaSet|} \times \sum_{i=1}^{|ClSet^*|} [-p_{ij} \ln(p_{ij})] \right]$$

where p_{ij} is the probability that an XML Schema of a cluster Cl_j belongs to the class Cl_i^* .

Values of Precision, Recall, and F-Measure fall in the real interval $[0, 1]$, whereas values of Overall vary between $-\infty$ and 1. The higher the value of these measures is, the better the accuracy of the approach into examination will be. Finally, values of Entropy belong to the real value $[0.. \ln(|ClSet^*|)]$; the lower the Entropy is, the purer the produced clusters will be.

Internal measures. Two interesting internal measures for evaluating clustering techniques are:

- *Uncoupling Degree*. This measure has been derived from the *coupling bound* measure introduced in [20]. Specifically, let Cl_i and Cl_j be two clusters and let τ be a number in the real interval $[0, 1]$; we define the set CU_{ij}^τ of the τ -uncoupled pairs between Cl_i and Cl_j as: $CU_{ij}^\tau = \{\langle S_a, S_b \rangle \mid S_a \in Cl_i, S_b \in Cl_j, d(S_a, S_b) \geq \tau\}$, where $d(S_a, S_b)$ represents the distance (i.e., the dissimilarity) between S_a and S_b ; in our approach it is computed by means of the formula introduced in Section 2.2.

The τ -Uncoupling Degree Unc_{ij}^τ between two clusters Cl_i and Cl_j is defined as the ratio between the τ -uncoupled pairs relative to Cl_i and Cl_j and the total number of possible pairs relative to Cl_i and Cl_j ; in other words, $Unc_{ij}^\tau = \frac{|CU_{ij}^\tau|}{|Cl_i| \times |Cl_j|}$.

Finally, the *Uncoupling Degree* U^τ is defined as $U^\tau = \min_{\substack{1 \leq i, j \leq |ClSet| \\ i \neq j}} Unc_{ij}^\tau$. U^τ belongs to the real interval $[0, 1]$ and measures the capability of a clustering algorithm to produce sufficiently separated clusters; given a value of τ , the higher U^τ is, the higher the separation between clusters will be.

- *Cohesiveness Degree*. This measure has been derived from the *cohesiveness* parameter introduced in [20]. Specifically, given a cluster Cl_i and a real number $\tau \in [0, 1]$, we define the set of τ -cohesive pairs as $CC_i^\tau = \{\langle S_a, S_b \rangle \mid S_a, S_b \in Cl_i, S_a \neq S_b, d(S_a, S_b) \leq \tau\}$.

Table 2. Precision, Recall, and Entropy of our approach for various values of α , β and γ

α	β	γ	K-MEANS			EM			FARTHEST FIRST TRAVERSAL		
			Precision	Recall	Entropy	Precision	Recall	Entropy	Precision	Recall	Entropy
0.9	0.8	0.8	0.80	0.91	0.37	0.88	0.92	0.32	0.88	0.87	0.35
0.9	0.8	0.2	0.81	0.93	0.34	0.90	0.93	0.30	0.90	0.88	0.33
0.8	0.6	0.6	0.83	0.94	0.32	0.91	0.95	0.26	0.91	0.90	0.30
0.8	0.6	0.4	0.85	0.96	0.30	0.94	0.96	0.24	0.93	0.92	0.28
0.8	0.4	0.5	0.82	0.91	0.35	0.91	0.92	0.27	0.89	0.87	0.34
0.8	0.4	0.3	0.83	0.93	0.33	0.92	0.94	0.25	0.91	0.90	0.31
0.6	0.4	0.7	0.81	0.90	0.37	0.88	0.90	0.28	0.88	0.89	0.36
0.6	0.2	0.6	0.80	0.88	0.40	0.86	0.89	0.31	0.86	0.85	0.38

The τ -Cohesiveness Degree of a cluster Cl_i is defined as the ratio between the number of τ -cohesive pairs and the total number of pairs of XML Schemas in Cl_i ; specifically, $Cohes_i^\tau = \frac{|CC_i^\tau|}{|Cl_i| \times (|Cl_i|-1)}$.

Finally, the *Cohesiveness Degree* C^τ is defined as $C^\tau = \min_{1 \leq i \leq |ClSet|} Cohes_i^\tau$. C^τ belongs to the real interval $[0, 1]$ and measures the capability of an algorithm to produce cohesive clusters, i.e., clusters composed by very “similar” XML Schemas.

As pointed out in [20], a clustering algorithm should produce very cohesive and sufficiently uncoupled clusters; therefore, the higher the values of internal measures are, the better the performance of the algorithm will be.

3.3 Tuning of the Parameters Exploited by Our Approach

Our approach exploits some parameters (see Section 2.2); therefore, before carrying out any test, we had to experimentally find the values to be associated with them for guaranteeing the best accuracy. In order to carry out such an evaluation, we have applied K-Means, EM, and Farthest First Traversal on the dissimilarity matrixes constructed by our approach and we have considered various values of the parameters to tune; after this, we have computed Precision, Recall, and Entropy on returned clusters.

Table 2 shows the values of these measures for some of the combinations we have considered. At the end of these tests we have found that our approach shows the best results for $\alpha = 0.8$, $\beta = 0.6$ and $\gamma = 0.4$.

The results we have obtained for α and β confirm our reasoning expressed in Section 2.2, when we say that synonymies are more important than overlappings and hyponymies/hypernyms; however, the quite high value of β shows that also these last kinds of properties play a sufficiently important role in characterizing the semantics of a concept. Analogously, the results we have obtained for γ confirm our reasoning expressed in Section 2.2, when we say that complex elements are more characterizing than simple elements and attributes in determining the semantics of a concept, even if, in any case, these last play an important role.

3.4 Evaluation of the Impact of Our Dissimilarity Matrix Computation Approach on the Clustering Quality

The quality of results produced by any clustering algorithm strongly depends on the dissimilarity matrix received in input, since it summarizes the relationships existing among the objects into examination. Clearly, a sophisticated approach for the calculation of the dissimilarity matrix causes an increase of the computational cost, on one hand, but it allows an improvement of the accuracy of obtained results, on the other hand.

Since our approach for the computation of the dissimilarity matrix is quite complex, we have planned to quantify the improvement it produces on the result accuracy w.r.t. an approach that takes into account the semantics of the involved Schemas in a simpler way. The “simplified” definition of the XML Schema distance exploited in this test (called d_S in the following) considers only the fraction of the dissimilarity properties existing between them. Specifically, d_S is defined as:

$$d_S(S_i, S_j) = 1 - \frac{|sim(S_i)| + |sim(S_j)|}{|XCompSet(S_i)| + |XCompSet(S_j)|}$$

where $sim(S_i)$, (resp., $sim(S_j)$) indicates the set of x-components of S_i (resp., S_j) involved in at least one synonymy with an x-component of S_j (resp., S_i).

It is worth pointing out that this dissimilarity measure is really used in the literature; moreover, we observe that it is not a metrics; for this reason, in this test, we have adopted Farthest First Traversal as clustering algorithm, since it does not necessarily need a metric space.

In this test we have performed two analyses, devoted to consider external and internal measures, respectively.

Analysis of the external measures. In a first series of experiments we have compared the values of the external measures obtained by applying our approach and the “simplified” one. Table 3 shows the obtained results; from its examination we deduce that our approach allows a substantial improvement on the quality of results; specifically, if compared with the “simplified” one, Precision increases of 19%, Recall improves of 12%, F-Measures increases of 14%, Overall improves of 42% and Entropy decreases of 18%.

Analysis of the internal measures. The computation of the internal measures depends on the parameter τ specifying when two XML Schemas can be

Table 3. Comparison of the accuracy of our approach w.r.t. the accuracy of the “simplified” one

Measure	Precision	Recall	F-Measure	Overall	Entropy
Our Approach	0.93	0.92	0.91	0.84	0.28
“Simplified” Approach	0.78	0.82	0.80	0.59	0.34

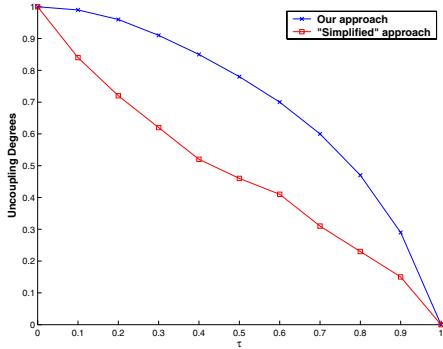


Fig. 1. Variation of the Uncoupling Degree against τ , obtained by exploiting our approach and the “simplified” one

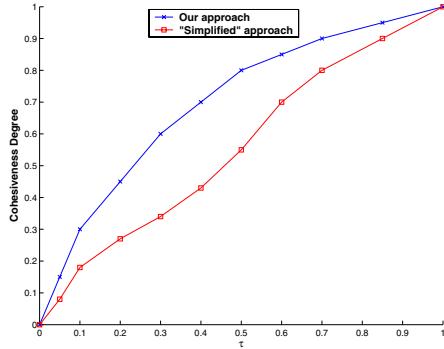


Fig. 2. Variation of the Cohesiveness Degree against τ , obtained by exploiting our approach and the “simplified” one

considered uncoupled or cohesive (see Section 3.2). In our analysis we have considered various values of τ and, for each of them, we have computed the Uncoupling and the Cohesiveness Degrees, obtained by exploiting our approach and the “simplified” one. Figures 1 and 2 show the corresponding results. Also in this case the improvement we have obtained is significant; in fact, the maximum increase of the Uncoupling Degree (resp., the Cohesiveness Degree) determined by exploiting our approach is obtained for $\tau = 0.40$ (resp., $\tau = 0.50$) and is equal to 0.33 (resp., 0.27).

3.5 Analysis of the Robustness of Our Approach

As pointed out in Section 2.2, our approach is based on the interschema properties stored in *IPD*. In order to measure its robustness against errors in *IPD*, we have carried out some variations in the correct dictionary and, for each of them, we have computed the values of the external measures returned by it. This analysis is important because, if the number of involved XML Schemas is high, it is compulsory to compute *IPD* (semi-)automatically; clearly, (semi-)automatic techniques are more error-prone than manual ones.

The variations we have performed in the correct dictionary are: (a) the removal of 10% of correct properties; (b) the removal of 20% of correct properties; (c) the removal of 30% of correct properties; (d) the removal of 50% of correct properties; (e) the insertion of 10% of wrong properties; (f) the insertion of 20% of wrong properties; (g) the insertion of 30% of wrong properties; (h) the insertion of 50% of wrong properties. The clustering algorithm we have used in this experiment was *EM* because it showed the highest accuracy in the previous tests.

Table 4 illustrates obtained results. From its analysis we can observe that our approach is quite robust against errors present in *IPD*; however, at the same time, it shows a good sensitivity against these errors since, if the correct

Table 4. Variations of the external measures in presence of errors in *IPD*

Case	Precision	Recall	F-Measure	Overall	Entropy
No error	0.94	0.96	0.94	0.86	0.24
(a)	0.94	0.87	0.89	0.80	0.25
(b)	0.93	0.78	0.84	0.71	0.26
(c)	0.92	0.69	0.78	0.61	0.28
(d)	0.92	0.62	0.73	0.55	0.30
(e)	0.85	0.94	0.88	0.75	0.28
(f)	0.78	0.90	0.83	0.63	0.33
(g)	0.70	0.87	0.77	0.50	0.37
(h)	0.63	0.84	0.71	0.38	0.43

properties that are removed or the wrong properties that are added are excessive, the accuracy of results significantly decreases.

4 Related Work

In this section we compare our approach with other related ones already presented in the literature.

Approach of [19]. In [19] an approach for performing the clustering of DTDs is proposed. It operates as follows: first it applies any clustering algorithm for grouping elements of the involved DTDs in a set of clusters. After this, it creates an array for each DTD, having a component for each cluster; the i^{th} component of the array indicates how many elements of the corresponding DTD belong to the i^{th} cluster. Finally, it applies any clustering algorithm on the set of constructed arrays.

There are some similarities between our approach and that described in [19]. Specifically: (i) both of them construct an *array-based* representation of the involved Schemas that, next, is provided in input to a clustering algorithm; (ii) both of them have been specifically conceived for XML.

The main differences between the two approaches are the following: (i) in [19] the computation of the similarity between two DTDs privileges their *structural* properties (i.e., the hierarchical organization of the corresponding elements); on the contrary, our approach considers interschema properties, that define a *semantic* information; (ii) the clustering activity performed during the first phase allows the approach described in [19] to carry out a preliminary reduction of the involved elements; this feature is not present in our approach; however, errors possibly occurring during this initial clustering activity can negatively influence the final results.

XClust. In [14] the system *XClust*, defining a DTD clustering technique as a part of a more complex DTD integration approach, is proposed. In *XClust* each DTD is modelled by means of a tree; this representation allows the definition of a *similarity measure* for each pair of elements belonging to different DTDs; these measures are, then, exploited for computing the *similarity degree* of two DTDs. Once the similarity degree associated with each pair of available DTDs has been computed, a hierarchical clustering algorithm is applied.

The main similarities between our approach and *XClust* are the following: *(i)* both of them have been specifically conceived for XML; *(ii)* both of them operate on the intensional component of the available information sources.

As for differences between the two approaches we observe that: *(i)* *XClust* considers only synonymies and does not take into account hyponymies and overlappings; *(ii)* *XClust* exploits various thresholds and weights for computing the similarity of two DTDs; as a consequence, it produces accurate results but requires quite a complex tuning activity.

Approach of [12]. In [12] an approach for clustering structured information sources present in the Web is proposed. It assumes the existence, for each application domain, of a *hidden model* containing a finite vocabulary of attributes; this assumption allows schemas to be clustered by means of a specific algorithm called MD (*Model Differentiation*).

The main similarities between our approach and that described in [12] are the following: *(i)* both of them define a suitable mechanism for representing involved sources; *(ii)* both of them exploit semantic information; specifically, our approach uses interschema properties whereas the approach of [12] considers the hidden model.

The main differences between the two approaches are the following: *(i)* the approach presented in [12] requires a deep analysis of the extensional component of the involved information sources; this analysis produces very satisfactory results but requires a significant pre-processing phase for constructing, among others, the hidden model; *(ii)* the approach proposed in [12] has been specifically conceived for analyzing structured information sources present in the Web whereas our approach is specialized for XML Schemas.

Approach of [16]. In [16] an approach for clustering XML documents is described. It models the available documents by means of *ordered trees* and exploits a dynamic programming algorithm for defining a similarity measure for them. Finally, it uses a hierarchical clustering algorithm to group documents into homogeneous classes.

There exist some similarities between our approach and that described in [16]. Specifically: *(i)* both of them propose a suitable model for representing involved information sources; in our case this model has a “vectorial” nature whereas, in the approach of [16], it is based on trees; *(ii)* both of them are *flexible*, in the sense that they allow the exploitation of any clustering algorithm.

As for the main differences between the two approaches, we observe that: *(i)* for computing the similarity degree among the involved sources, the approach described in [16] considers the structural information whereas our approach exploits the semantic one; *(ii)* our approach focuses on XML Schemas whereas the approach of [16] has been conceived to operate on XML documents.

Approach of [6]. In [6] an approach for clustering XML documents is proposed. It represents available documents by means of ordered trees and measures their similarity by means of a dynamic programming algorithm; after this, it con-

structs a labelled graph G , whose nodes represent XML documents and whose arcs denote the corresponding similarity degrees; finally, it applies the Prim algorithm for partitioning the set of nodes of G and associates a cluster with each partition.

There are some similarities between our approach and that presented in [6]; specifically, (i) both of them have been conceived for XML; (ii) both of them define a suitable mechanism for representing information sources; this is “vectorial”-based in our approach and tree-based in the approach of [6].

The main differences existing between the two approaches are the following: (i) in order to compute the document similarity, the approach described in [6] exploits the structural information of involved sources, whereas our approach considers the semantic one; (ii) the approach illustrated in [6] operates on the extensional component of the information sources into consideration, whereas our approach works on the intensional one.

Approach of [5]. [5] presents an approach for clustering XML documents on the basis of their structural similarities. This approach represents each document by means of a tree; then, it applies *tree matching* algorithms for identifying the structural similarities existing among the available trees. In this way, it is possible to partition available documents into homogeneous classes and, then, to define, for each class, a tree (called *XML cluster representative*) summarizing the main characteristics of the trees belonging to it. This partitioning is, finally, refined by applying a suitable hierarchical clustering algorithm called **XRep**.

There exist some similarities between our approach and that described in [5]. Specifically: (i) both of them propose a suitable formalism for representing involved information sources; in our approach it has a “vectorial” nature whereas, in the approach of [5], it is tree-based; (ii) both of them operate on XML sources.

The main differences between the two approaches are the following: (i) the approach of [5] exploits structural information for computing similarities existing between two XML documents whereas, for the same purpose, our approach exploits semantic information; (ii) the approach of [5] is quite sophisticated; as a consequence, it produces very refined results but requires quite a complex pre-processing phase; on the contrary, our approach is lighter, even if the results it obtains are satisfactory.

5 Conclusions

In this paper we have presented an approach that exploits interschema properties for clustering semantically heterogeneous XML Schemas. We have seen that our approach takes the semantics of involved Schemas into account and can be easily integrated with most of the clustering algorithms already proposed in the literature.

After a technical description of our approach, we have shown various experimental results that we have obtained by applying it to a large number of semantically heterogeneous XML Schemas. Finally, we have presented a comparison between it and other related approaches previously proposed in the literature.

In our opinion the approach presented here could be improved in several directions. Specifically, we plan to further refine the technique for the computation of the dissimilarity matrix by taking other interschema properties into account. In addition, we would like to exploit our approach as the core of new methodologies for constructing multi-thematic portals, for guiding users in their navigation on the Web and, finally, for grouping the users of a service on the basis of both their profile and their past behaviour.

Acknowledgments

The authors thank Giuseppe Meduri for his contribution to the implementation of the proposed approach.

References

1. F. Beil, M. Ester, and X. Xu. Frequent term-based text clustering. In *Proc. of the International Conference on Knowledge Discovery and Data Mining (KDD'02)*, pages 436–442, Edmonton, Alberta, Canada, 2002. ACM Press.
2. S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, 1999.
3. F. Buccafurri, D. Rosaci, G.M.L. Sarnè, and D. Ursino. An agent-based hierarchical clustering approach for e-commerce environments. In *Proc. of International Conference on Electronic Commerce and Web Technologies (EC-Web 2002)*, pages 109–118, Aix-en-Provence, France, 2002. Lecture Notes in Computer Science, Springer-Verlag.
4. S. Castano, V. De Antonellis, and S. De Capitani di Vimercati. Global viewing of heterogeneous data sources. *IEEE Transactions on Data and Knowledge Engineering*, 13(2):277–297, 2001.
5. G. Costa, G. Manco, R. Ortale, and A. Tagarelli. A tree-based approach to clustering XML documents by structure. In *Proc. of the European Conference on Principles of Knowledge Discovery in Databases (PKDD 2004)*, pages 137–148, Pisa, Italy, 2004. Springer.
6. T. Dalamagas, T. Cheng, K. Winkel, and T.K. Sellis. Clustering XML documents using structural summaries. In *Proc. of the International Workshop on Clustering Information Over the Web (ClustWeb 2004)*, pages 547–556, Heraklion, Crete, Greece, 2004. Lecture Notes in Computer Science, Springer.
7. P. De Meo, G. Quattrone, G. Terracina, and D. Ursino. “Almost automatic” and semantic integration of XML Schemas at various “severity levels”. In *Proc. of the International Conference on Cooperative Information Systems (CoopIS 2003)*, pages 4–21, Taormina, Italy, 2003. Lecture Notes in Computer Science, Springer.
8. P. De Meo, G. Quattrone, G. Terracina, and D. Ursino. Extraction of synonymies, hyponymies, overlappings and homonymies from XML Schemas at various “severity” levels. In *Proc. of the International Database Engineering and Applications Symposium (IDEAS 2004)*, pages 389–394, Coimbra, Portugal, 2004. IEEE Computer Society.

9. A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society Series B*, 30(1):1–38, 1977.
10. P. Fankhauser, M. Kracker, and E.J. Neuhold. Semantic vs. structural resemblance of classes. *ACM SIGMOD RECORD*, 20(4):59–63, 1991.
11. J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2001.
12. B. He, T. Tao, and K. Chen-Chuan Chang. Organizing structured Web sources by query schemas: a clustering approach. In *Proc. of the ACM International Conference on Information and Knowledge Management (CIKM 2004)*, pages 22–31, Washington, Columbia, USA, 2004. ACM Press.
13. D.S. Hochbaum and D.B. Shmoys. A best possible heuristic for the k-center problem. *International Journal on Digital Libraries*, 10(2):180–184, 1985.
14. M.L. Lee, L.H. Yang, W. Hsu, and X. Yang. XClust: clustering XML schemas for effective integration. In *Proc. of the ACM International Conference on Information and Knowledge Management (CIKM 2002)*, pages 292–299, McLean, Virginia, USA, 2002. ACM Press.
15. J.B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the International Symposium on Mathematics, Statistics and Probability*, pages 281–297, Berkeley, California, USA, 1967. University of California Press.
16. A. Nierman and H.V. Jagadish. Evaluating structural similarity in XML documents. In *Proc. of the International Workshop on the Web and Databases (WebDB 2002)*, pages 61–66, Madison, Wisconsin, USA, 2002.
17. L. Palopoli, D. Saccà, G. Terracina, and D. Ursino. Uniform techniques for deriving similarities of objects and subschemes in heterogeneous databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):271–294, 2003.
18. K. Passi, L. Lane, S.K. Madria, B.C. Sakamuri, M.K. Mohania, and S.S. Bhowmick. A model for XML Schema integration. In *Proc. of the International Conference on E-Commerce and Web Technologies (EC-Web 2002)*, pages 193–202, Aix-en-Provence, France, 2002. Lecture Notes in Computer Science, Springer.
19. W. Qian, L. Zhang, Y. Liang, H. Qian, and W. Jin. A two-level method for clustering DTDs. In *Proc. of the International Conference on Web-Age Information Management (WAIM'00)*, pages 41–52, Shanghai, China, 2000. Lecture Notes in Computer Science, Springer.
20. Y. Qian and K. Zhang. A customizable hybrid approach to data clustering. In *Proc. of the International Symposium on Applied Computing (SAC'03)*, pages 485–489, Melbourne, Florida, USA, 2003. ACM Press.
21. E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.
22. C.J. Van Rijsbergen. *Information Retrieval*. Butterworth, 1979.
23. I.H. Witten and E. Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, California, USA, 2000.
24. L. Xu and M. I. Jordan. On convergence properties of the em algorithm for gaussian mixtures. *Neural Computation*, 8(1):129–151, 1996.

Semantic Schema Matching*

Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich

University of Trento, Povo, Trento, Italy

{fausto, pavel, yatskevi}@dit.unitn.it

Abstract. We view *match* as an operator that takes two graph-like structures (e.g., XML schemas) and produces a mapping between the nodes of these graphs that correspond semantically to each other. *Semantic schema matching* is based on the two ideas: (i) we discover mappings by computing *semantic relations* (e.g., equivalence, more general); (ii) we determine semantic relations by analyzing the *meaning* (concepts, not labels) which is codified in the elements and the structures of schemas. In this paper we present basic and optimized algorithms for semantic schema matching, and we discuss their implementation within the S-Match system. We also validate the approach and evaluate S-Match against three state of the art matching systems. The results look promising, in particular for what concerns quality and performance.

1 Introduction

Match is a critical operator in many well-known metadata intensive applications, such as schema/classification/ontology integration, data warehouses, e-commerce, semantic web, etc. The match operator takes two graph-like structures and produces a mapping between the nodes of the graphs that correspond semantically to each other.

Many diverse solutions of match have been proposed so far, for example [2, 5, 7, 8, 10, 16, 17, 19]. We focus on a schema-based solution, namely a matching system exploiting only the schema information, thus not considering instances. We follow a novel approach called *semantic matching* [11]. This approach is based on the two key ideas. The first is that we calculate mappings between schema elements by computing *semantic relations* (e.g., equivalence, more generality, disjointness), instead of computing coefficients rating match quality in the [0,1] range, as it is the case in the most previous approaches, see, for example, [8, 17, 19]. The second idea is that we determine semantic relations by analyzing the *meaning* (concepts, not labels) which is codified in the elements and the structures of schemas. In particular, labels at nodes, written in natural language, are translated into propositional formulas which explicitly codify the label's intended meaning. This allows us to translate the matching problem into a propositional unsatisfiability problem, which can then be efficiently resolved using (sound and complete) state of the art propositional satisfiability (SAT) deciders, e.g., [4].

* We are grateful to John Mylopoulos and anonymous reviewers for their insightful comments on the semifinal version of the paper.

A vision for semantic matching approach and its implementation were reported in [11–13]. In contrast to that work which have been focused only on matching classifications or light-weight ontologies, this paper also considers matching XML schemas. It elaborates in more detail the element level and the structure level matching algorithms, providing a complete account of the approach. In particular, the main contributions are: (i) a new schema matching algorithm, which builds on the advances of the previous solutions at the element level by providing a library of element level matchers, and guarantees correctness and completeness of its results at the structure level; (ii) an extension of the semantic matching approach for handling attributes; (iii) the quality and performance evaluation of the implemented system called S-Match against other state of the art systems, which proves empirically the benefits of our approach.

The rest of the paper is organized as follows. Section 2 provides the related work. A basic version of the matching algorithm is articulated in its four macro steps in Section 3, while its optimizations are reported in Section 4. Section 5 discusses semantic matching with attributes. Section 6 presents a comparative evaluation. Finally, Section 7 reports conclusions and discusses the future work.

2 Related Work

At present, there exists a line of semi-automated schema matching systems, see, for instance [2, 7, 8, 10, 16, 17, 19]. A good survey and a classification of matching approaches up to 2001 is provided in [24], while an extension of its schema-based part and a user-centric classification of matching systems is provided in [25].

In particular, for individual matchers, [25] introduces the following criteria which allow for detailing further (with respect to [24]), the element and structure level of matching: *syntactic techniques* (these interpret their input as a function of their sole structures following some clearly stated algorithms, e.g., iterative fix point computation for matching graphs), *external techniques* (these exploit external resources of a domain and common knowledge, e.g., WordNet[21]), and *semantic techniques* (these use formal semantics, e.g., model-theoretic semantics, in order to interpret the input and justify their results).

The distinction between the hybrid and composite matching algorithms of [24] is useful from an architectural perspective. [25] extends this work by taking into account how the systems can be distinguished in the matter of considering the mappings and the matching task, thus representing the end-user perspective. In this respect, the following criteria are proposed: *mappings as solutions* (these systems consider the matching problem as an optimization problem and the mapping is a solution to it, e.g., [9, 19]); *mappings as theorems* (these systems rely on semantics and require the mapping to satisfy it, e.g., the approach proposed in this paper); *mappings as likeness clues* (these systems produce only reasonable indications to a user for selecting the mappings, e.g., [8, 17]).

Let us consider some recent schema-based state of the art systems in light of the above criteria.

Rondo. The Similarity Flooding (SF) [19] approach, as implemented in Rondo [20], utilizes a hybrid matching algorithm based on the ideas of similarity propagation. Schemas are presented as directed labeled graphs. The algorithm exploits only syntactic techniques at the element and structure level. It starts from the string-based comparison (common prefixes, suffixes tests) of the node’s labels to obtain an initial mapping which is further refined within the fix-point computation. SF considers the mappings as a solution to a clearly stated optimization problem.

Cupid. Cupid [17] implements a hybrid matching algorithm comprising syntactic techniques at the element (e.g., common prefixes, suffixes tests) and structure level (e.g., tree matching weighted by leaves). It also exploits external resources, in particular, a precompiled thesaurus. Cupid falls into the mappings as likeness clues category.

COMA. COMA [8] is a composite schema matching system which exploits syntactic and external techniques. It provides a library of matching algorithms; a framework for combining obtained results, and a platform for the evaluation of the effectiveness of the different matchers. The matching library is extensible, it contains 6 elementary matchers, 5 hybrid matchers, and one reuse-oriented matcher. Most of them implement string-based techniques (affix, n-gram, edit distance, etc.); others share techniques with Cupid (tree matching weighted by leaves, thesauri look-up, etc.); reuse-oriented is a completely novel matcher, which tries to reuse previously obtained results for entire new schemas or for its fragments. Distinct features of COMA with respect to Cupid, are a more flexible architecture and a possibility of performing iterations in the matching process. COMA falls into the mappings as likeness clues category.

3 Semantic Matching

We focus on tree-like structures, e.g., XML schemas. Real-world schemas are seldom trees, however, there are (optimized) techniques, transforming a graph representation of a schema into a tree representation, e.g., the graph-to-tree operator of Protoplasm [3].

We call *concept of a label* the propositional formula which stands for the set of data instances that one would classify under a label it encodes. We call *concept at a node* the propositional formula which represents the set of data instances which one would classify under a node, given that it has a certain label and that it is in a certain position in a tree.

The semantic matching approach can discover the following semantic relations between the concepts of nodes of the two schemas: *equivalence* ($=$); *more general* (\supseteq); *less general* (\subseteq); *disjointness* (\perp). When none of the relations holds, the special *idk* (I don’t know) relation is returned. The relations are ordered according to decreasing binding strength, i.e., from the strongest ($=$) to the weakest (*idk*), with more general and less general relations having equal binding power. The semantics of the above relations are the obvious set-theoretic semantics.

A *mapping element* is a 4-tuple $\langle ID_{ij}, n1_i, n2_j, R \rangle$, $i=1,\dots,N1; j=1,\dots,N2$; where ID_{ij} is a unique identifier of the given mapping element; $n1_i$ is the i -th node of the first tree, $N1$ is the number of nodes in the first tree; $n2_j$ is the j -th node of the second tree, $N2$ is the number of nodes in the second tree; and R specifies a semantic relation which may hold between the *concepts of nodes* $n1_i$ and $n2_j$. *Semantic matching* can then be defined as the following problem: given two trees $T1, T2$ compute the $N1 \times N2$ mapping elements $\langle ID_{ij}, n1_i, n2_j, R' \rangle$, with $n1_i \in T1, i=1,\dots,N1, n2_j \in T2, j=1,\dots,N2$ and R' the strongest semantic relation holding between the concepts of nodes $n1_i, n2_j$.

3.1 The Tree Matching Algorithm

We summarize the algorithm for semantic schema matching via a running example. We consider the two simple XML schemas shown in Figure 1.

Let us introduce some notation (see also Figure 1). Numbers are the unique identifiers of nodes. We use " C " for concepts of labels and concepts at nodes. Also we use " $C1$ " and " $C2$ " to distinguish between concepts of labels and concepts at nodes in tree 1 and tree 2 respectively. Thus, in A1, $C1_{Photo_and_Cameras}$ and $C1_3$ are, respectively, the concept of the label *Photo_and_Cameras* and the concept at node 3.

The algorithm takes as input two schemas and computes as output a set of mapping elements in four macro steps. The first two steps represent the pre-processing phase, while the third and the fourth steps are the element level and structure level matching respectively.

Step 1. For all labels L in the two trees, compute *concepts of labels*. We think of labels at nodes as concise descriptions of the data that is stored under the nodes. We compute the meaning of a label at a node by taking as input a label, by analyzing its real-world semantics, and by returning as output a concept

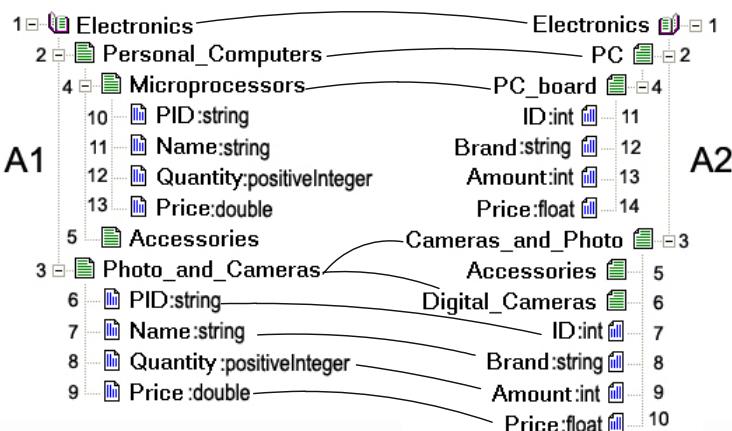


Fig. 1. Two XML schemas and some of the mappings

of the label, C_L . Thus, for example, by writing $C_{Cameras_and_Photo}$ we move from the natural language ambiguous label *Cameras_and_Photo* to the concept $C_{Cameras_and_Photo}$, which codifies explicitly its intended meaning, namely the data which is about cameras and photo.

Technically, we codify concepts of labels as propositional logical formulas. First, we chunk labels into *tokens*, e.g., *Photo_and_Cameras* \rightarrow $\langle photo, and, cameras \rangle$; and then, we extract *lemmas* from the tokens, e.g., *cameras* \rightarrow *camera*. Atomic formulas are WordNet [21] *senses* of lemmas obtained from single words (e.g., *cameras*) or multiwords (e.g., *digital cameras*). Complex formulas are built by combining atomic formulas using the connectives of set theory. For example, $C_{2Cameras_and_Photo} = \langle Cameras, senses_{WN\#2} \rangle \sqcup \langle Photo, senses_{WN\#1} \rangle$, where $senses_{WN\#2}$ is taken to be disjunction of the two senses that WordNet attaches to *Cameras*, and similarly for *Photo*. Notice that the natural language conjunction "and" has been translated into the logical disjunction " \sqcup ".

From now on, to simplify the presentation, we assume that the propositional formula encoding the concept of label is the label itself. We use numbers "1" and "2" as subscripts to distinguish between trees in which the given concept of label occurs. Thus, for example, $C_{Cameras_and_Photo_2}$ is a notational equivalent of $C_{2Cameras_and_Photo}$.

Step 2. For all nodes N in the two trees, compute *concepts of nodes*. In this step we analyze the meaning of the positions that the labels at nodes have in a tree. By doing this we *extend* concepts of labels to *concepts of nodes*, C_N . This is required to capture the knowledge residing in the structure of a tree, namely the context in which the given concept at label occurs. For example, in A2, when we write C_6 we mean the concept describing all the data instances of the electronic photography products which are digital cameras.

Technically, concepts of nodes are written in the same propositional logical language as concepts of labels. XML schemas are hierarchical structures where the path from the root to a node uniquely identifies that node (and also its meaning). Thus, following an *access criterion* semantics [14], the logical formula for a concept at node is defined as a conjunction of concepts of labels located in the path from the given node to the root. For example, $C_{26} = Electronics_2 \sqcap Cameras_and_Photo_2 \sqcap Digital_Cameras_2$.

Step 3. For all pairs of labels in the two trees, compute *relations among concepts of labels*. Relations between concepts of labels are computed with the help of a library of element level semantic matchers. These matchers take as input two atomic concepts of labels and produce as output a semantic relation between them. Some of the them are re-implementations of the well-known matchers used in Cupid and COMA. The most important difference is that our matchers return a semantic relation (e.g., $=$, \sqsubseteq , \sqsubseteq), rather an affinity level in the $[0,1]$ range, although sometimes using customizable thresholds.

The element level semantic matchers are briefly summarized in Table 1. The first column contains the names of the matchers. The second column lists the order in which they are executed. The third column introduces the matcher's

Table 1. Element level semantic matchers

Matcher name	Execution order	Approximation level	Matcher type	Schema info
Prefix	2	2	String-based	Labels
Suffix	3	2	String-based	Labels
Edit distance	4	2	String-based	Labels
Ngram	5	2	String-based	Labels
Text corpus	12	3	String-based	Labels + corpus
WordNet	1	1	Sense-based	WordNet senses
Hierarchy distance	6	3	Sense-based	WordNet senses
WordNet gloss	7	3	Gloss-based	WordNet senses
Extended WordNet gloss	8	3	Gloss-based	WordNet senses
Gloss comparison	9	3	Gloss-based	WordNet senses
Extended gloss comparison	10	3	Gloss-based	WordNet senses
Extended semantic gloss comparison	11	3	Gloss-based	WordNet senses

approximation level. The relations produced by a matcher with the first approximation level are always correct. For example, *name* \sqsupseteq *brand* returned by *WordNet*. In fact, according to *WordNet* *name* is a hypernym (superordinate word) to *brand*. Notice that in *WordNet* *name* has 15 senses and *brand* has 9 senses. We use some sense filtering techniques to discard the irrelevant senses for the given context, see [18] for details. The relations produced by a matcher with the second approximation level are likely to be correct (e.g., *net* = *network*, but *hot* = *hotel* by *Suffix*). The relations produced by a matcher with the third approximation level depend heavily on the context of the matching task (e.g., *cat* = *dog* by *Extended gloss comparison* in the sense that they are both *pets*). Notice that matchers are executed following the order of increasing approximation. The fourth column reports the matcher's type, while the fifth column describes the matcher's input.

As from Table 1, we have three main categories of matchers. *String-based* matchers have two labels as input (with exception of *Text corpus* which takes in input also a text corpus). These compute only equivalence relations (e.g., equivalence holds if the weighted *distance* between the input strings is lower than a threshold). *Sense-based* matchers have two *WordNet* senses in input. The *WordNet* matcher computes equivalence, more/less general, and disjointness relations; while *Hierarchy distance* computes only the equivalence relation. *Gloss-based* matchers also have two *WordNet* senses as input, however they exploit techniques based on comparison of textual definitions (*glosses*) of the words whose senses are taken in input. These compute, depending on a particular matcher, the equivalence, more/less general relations. The result of step 3 is a matrix of the relations holding between concepts of labels. A part of this matrix for the example of Figure 1 is shown in Table 2.

Table 2. The matrix of semantic relations holding between concepts of labels

	<i>Cameras</i> ₂	<i>Photo</i> ₂	<i>Digital-Cameras</i> ₂
<i>Photo</i> ₁	<i>idk</i>	=	<i>idk</i>
<i>Cameras</i> ₁	=	<i>idk</i>	\sqsupseteq

Step 4. For all pairs of nodes in the two trees, compute relations among concepts of nodes. During this step, we initially reformulate the tree matching problem into a set of node matching problems (one problem for each pair of nodes). Finally, we translate each node matching problem into a propositional validity problem. Let us discuss in detail the tree matching algorithm, see Algorithm 1 for the pseudo-code.

Algorithm 1. The tree matching algorithm

```

1: Node: struct of
2:   int nodeId;
3:   String label;
4:   String cLabel;
5:   String cNode;
6: String[][] treeMatch(Tree of Nodes source, target)
7: Node sourceNode, targetNode;
8: String[][] cLabsMatrix, cNodesMatrix, relMatrix;
9: String axioms, context1, context2;
10: int i,j;
11: cLabsMatrix = fillCLabMatrix(source, target);
12: for each sourceNode ∈ source do
13:   i = getNodeID(sourceNode);
14:   context1 = getCnodeFormula(sourceNode);
15:   for each targetNode ∈ target do
16:     j = getNodeID(targetNode);
17:     context2 = getCnodeFormula(targetNode);
18:     relMatrix = extractRelMatrix(cLabMatrix, sourceNode, targetNode);
19:     axioms = mkAxioms(relMatrix);
20:     cNodesMatrix[i][j] = nodeMatch(axioms, context1, context2);
21:   end for
22: end for
23: return cNodesMatrix;
```

In line 6, the `treeMatch` function takes two trees of `Nodes` (`source` and `target`) in input. It starts from the element level matching. Thus, in line 11, the matrix of relations holding between concepts of labels (`cLabsMatrix`) is populated by the `fillCLabsMatrix` function which uses the library of element level matchers. We run two loops over all the nodes of `source` and `target` trees in lines 12-22 and 15-21 in order to formulate all our node matching problems. Then, for each node matching problem we take a pair of propositional formulas encoding concepts of nodes and relevant relations holding between concepts of labels using the `getCnodeFormula` and `extractRelMatrix` functions respectively. The former are memorized as `context1` and `context2` in lines 14 and 17. The latter are memorized in `relMatrix` in line 18. In order to reason about relations between concepts of nodes, we build the premises (`axioms`) in line 19. These are a conjunction of the concepts of labels which are related in `relMatrix`. For example, the task of matching $C1_3$ and $C2_6$, requires the following axioms: $(Electronics_1 = Electronics_2) \sqcap (Cameras_1 = Cameras_2) \sqcap (Photo_1 = Photo_2) \sqcap (Cameras_1 \sqsupset Digital_Cameras_2)$. Finally, in line 20, the relations holding between the concepts of nodes are calculated by `nodeMatch` and are reported in line 23 as a bidimensional array (`cNodesMatrix`). A part of this matrix for the example of Figure 1 is shown in Table 3.

Table 3. The matrix of semantic relations holding between concepts of nodes (the matching result)

	$C2_1$	$C2_2$	$C2_3$	$C2_4$	$C2_5$	$C2_6$
$C1_3$	\sqsubseteq	idk	$=$	idk	\sqsupseteq	\sqsupseteq

3.2 The Node Matching Algorithm

We translate the node matching problem into a propositional validity problem. Semantic relations are translated into propositional connectives using the rules described in Table 4 (second column). The criterion for determining whether a relation holds between concepts of nodes is the fact that it is entailed by the premises. Thus, we have to prove that the following formula:

$$axioms \longrightarrow rel(context_1, context_2) \quad (1)$$

is valid, namely that it is *true* for all the truth assignments of all the propositional variables occurring in it. *axioms*, *context*₁, and *context*₂ are as defined in the tree matching algorithm. *rel* is the semantic relation that we want to prove holding between *context*₁ and *context*₂. The algorithm checks for the validity of formula (1) by proving that its negation, i.e., formula (2), is unsatisfiable.

$$axioms \wedge \neg rel(context_1, context_2) \quad (2)$$

Table 4 (third column) describes how formula (2) is translated before testing each semantic relation. Notice that (2) is in Conjunctive Normal Form (CNF), namely it is a conjunction of disjunctions of atomic formulas. In this case we assume that atomic formulas never occur negated, following what is common practice in building labels of, e.g., XML schemas. Also, notice that $a = b$ iff both $a \sqsubseteq b$ and $b \sqsubseteq a$ hold, therefore we do not need to test the equivalence relation separately.

Table 4. The relationship between semantic relations and propositional formulas

$rel(a, b)$	Translation of $rel(a, b)$ into propositional logic	Translation of formula (2) into Conjunctive Normal Form
$a = b$	$a \leftrightarrow b$	N/A
$a \sqsubseteq b$	$a \rightarrow b$	$axioms \wedge context_1 \wedge \neg context_2$
$a \sqsupseteq b$	$b \rightarrow a$	$axioms \wedge context_2 \wedge \neg context_1$
$a \perp b$	$\neg(a \wedge b)$	$axioms \wedge context_1 \wedge context_2$

The pseudo-code of a basic solution for the node matching algorithm is provided in Algorithm 2. Let us analyze it in detail. In lines 110 and 140, the `node-Match` function constructs the formulas for testing the less general and more general relations. In lines 120 and 150, it converts them into CNF, while in lines 130 and 160, it checks formulas in CNF for unsatisfiability. If both relations hold, then the equivalence relation is returned (line 180). Finally, the same procedure

Algorithm 2. The node matching algorithm

```

100. String nodeMatch(String axioms, context1, context2)
110. String formula = And(axioms, context1, Not(context2));
120. String formulaInCNF = convertToCNF(formula);
130. boolean isLG = isUnsatisfiable(formulaInCNF);
140. formula = And(axioms, Not(context1), context2);
150. formulaInCNF = convertToCNF(formula);
160. boolean isMG = isUnsatisfiable(formulaInCNF);
170. if(isMG && isLG) then
180.   return "=";
190. endif
200. if (isLG) then
210.   return " $\sqsubseteq$ ";
220. endif
230. if (isMG) then
240.   return " $\sqsupseteq$ ";
250. endif
260. formula = And(axioms, context1, context2);
270. formulaInCNF = convertToCNF(formula);
280. boolean isOpposite = isUnsatisfiable(formulaInCNF);
290. if (isOpposite) then
300.   return " $\perp$ ";
310. else
320.   return "idk";
330. endif

```

is repeated for the disjointness relation. If all the tests fail, the idk relation is returned (line 320).

In order to check the unsatisfiability of a propositional formula in a basic version of our NodeMatch algorithm we use the standard DPLL-based SAT solver [4, 6]. From the example in Figure 1, trying to prove that $C2_6$ is less general than $C1_3$, requires constructing formula (3), which turns out to be unsatisfiable, and therefore, the less generality holds.

$$\begin{aligned}
& ((Electronics_1 \leftrightarrow Electronics_2) \wedge (Photo_1 \leftrightarrow Photo_2) \wedge \\
& (Cameras_1 \leftrightarrow Cameras_2) \wedge (Digital_Cameras_2 \rightarrow Cameras_1)) \wedge \\
& (Electronics_2 \wedge (Cameras_2 \vee Photo_2) \wedge Digital_Cameras_2) \wedge \neg \\
& (Electronics_1 \wedge (Photo_1 \vee Cameras_1))
\end{aligned} \tag{3}$$

4 Efficient Semantic Matching

In this section we present a set of optimizations for the node matching algorithm. In particular, we show, that when dealing with *conjunctive concepts at nodes*, i.e., the concept of node is a conjunction (e.g., $C1_2 = Electronics_1 \wedge Personal_Computers_1$), these node matching tasks can be solved in linear time. When we have *disjunctive concepts at nodes*, i.e., the concept of node contains both conjunctions and disjunctions in any order (e.g., $C2_6 = Electronics_2 \wedge (Cameras_2 \vee Photos_2) \wedge Digital_Cameras_2$), we use techniques avoiding the exponential space explosion which arises due to the conversion of disjunctive formulas into CNF. This modification is required since all state of the art SAT deciders take CNF formulas in input.

4.1 Conjunctive Concepts at Nodes

Let us make some observations with respect to Table 4. The first observation is that *axioms* remains the same for all the tests, and it contains only clauses with two variables. In the worst case, it contains $2 \cdot n_1 \cdot n_2$ clauses, where n_1 and n_2 are the number of atomic concepts of labels occurred in *context*₁ and *context*₂ respectively. The second observation is that the formulas for less and more generality tests are very similar and they differ only in the negated context formula (e.g., in the less generality test *context*₂ is negated). This means that formula (1) contains one clause with n_2 variables plus n_1 clauses with one variable. In the case of disjointness test *context*₁ and *context*₂ are not negated. Therefore, formula (1) contains $n_1 + n_2$ clauses with one variable. For lack of space, let us only consider tests for more/less general relations.

Less and more generality tests. Using the above observations, formula (1), with respect to the less/more generality tests, can be represented as follows:

$$\overbrace{\bigwedge_0^{n*m} (\neg A_s \vee B_t) \wedge \bigwedge_0^{n*m} (A_k \vee \neg B_l) \wedge \bigwedge_0^{n*m} (\neg A_p \vee \neg B_r)}^{\text{axioms}} \wedge \underbrace{\bigwedge_{i=1}^n A_i}_{\text{context}_1} \wedge \underbrace{\bigvee_{j=1}^m \neg B_j}_{\neg \text{context}_2} \quad (4)$$

where n is the number of variables in *context*₁, m is the number of variables in *context*₂. The A_i 's belong to *context*₁, and the B_j 's belong to *context*₂. s, k, p are in the [0..n] range, while t, l, r are in the [0..m] range. Axioms can be empty. Formula (4) is composed of clauses with one or two variables plus one clause with possibly more variables (the clause corresponding to the negated context). Notice that formula (4) is *Horn*, i.e., each clause contains at most one positive literal. Therefore, its satisfiability can be decided in linear time by the *unit resolution rule*. DPLL-based SAT solvers in this case require quadratic time. In order to understand how the linear time algorithm works, let us suppose that we want to check if $C1_4$ is less general than $C2_4$. Formula (4) in this case is as follows:

$$\begin{aligned} & ((\neg \mathbf{Electronics}_1 \vee \mathbf{Electronics}_2) \wedge (\mathbf{Electronics}_1 \vee \neg \mathbf{Electronics}_2) \wedge \\ & (\neg \mathbf{Personal_Computers}_1 \vee \mathbf{PC}_2) \wedge (\mathbf{Personal_Computers}_1 \vee \neg \mathbf{PC}_2) \wedge \\ & (\neg \mathbf{Microprocessors}_1 \vee \neg \mathbf{PC_board}_2)) \wedge \\ & \mathbf{Electronics}_1 \wedge \mathbf{Personal_Computers}_1 \wedge \mathbf{Microprocessors}_1 \wedge \\ & (\neg \mathbf{Electronics}_2 \vee \neg \mathbf{PC}_2 \vee \neg \mathbf{PC_board}_2) \end{aligned} \quad (5)$$

where the variables from *context*₁ are written in bold. First, we assign true to all the unit clauses occurring in (5) positively. Notice that these are all and only the clauses in *context*₁, namely, $\mathbf{Electronics}_1$, $\mathbf{Personal_Computers}_1$, and $\mathbf{Microprocessors}_1$. This allows us to discard the clauses where variables from *context*₁ occur positively, namely, $(\mathbf{Electronics}_1 \vee \neg \mathbf{Electronics}_2)$ and $(\mathbf{Personal_Computers}_1 \vee \neg \mathbf{PC}_2)$. Thus, the resulting formula is as follows:

$$\begin{aligned} & (\mathbf{Electronics}_2 \wedge \mathbf{PC}_2 \wedge \neg \mathbf{PC_board}_2) \wedge \\ & (\neg \mathbf{Electronics}_2 \vee \neg \mathbf{PC}_2 \vee \neg \mathbf{PC_board}_2) \end{aligned} \quad (6)$$

Formula (6) does not contain any variable from $context_1$. By assigning true to $Electronics_2$ and false to PC_board_2 we do not determine a contradiction, and therefore, (6) is satisfiable.

For formula (6) to be unsatisfiable, all the variables occurring in the negation of $context_2$, namely, $(\neg Electronics_2 \vee \neg PC_2 \vee \neg PC_board_2)$ should occur positively in the unit clauses obtained after resolving *axioms* with the unit clauses in $context_1$, namely, $Electronics_2$ and PC_2 . For this to happen, for any Bj there must be a clause of the form $\neg Ai \vee Bj$ in *axioms*. Formulas of the form $\neg Ai \vee Bj$ occur in (4) iff we have the axioms of type $Ai = Bj$ and $Ai \sqsubseteq Bj$. These considerations suggest the following algorithm for testing satisfiability:

- *Step 1.* Create an array of size m . Each entry in the array stands for one Bj in (4).
- *Step 2.* For each axiom of type $Ai = Bj$ and $Ai \sqsubseteq Bj$ mark the corresponding Bj .
- *Step 3.* If all the Bj ’s are marked, then the formula is unsatisfiable.

Thus, `nodeMatch` can be optimized by using Algorithm 3. The numbers on the left indicate where the new code must be positioned in Algorithm 2. `fastHornUnsatCheck` implements the three steps above. Step 1 is performed in lines 402 and 403. In lines 404-409, a loop on `axioms` implements Step 2. The final loop in lines 410-416 implements Step 3.

Algorithm 3. Optimizations: less/more generality tests

```

101. if (context1 and context2 are conjunctive) then
102.   isLG = fastHornUnsatCheck(context1, axioms, " $\sqsubseteq$ ");
103.   isMG = fastHornUnsatCheck(context2, axioms, " $\sqsupseteq$ ");
104. endif

401. boolean fastHornUnsatCheck(String context, axioms, rel)
402. int m = getNumOfVar(String context);
403. boolean array[m];
404. for each axiom ∈ axioms do
405.   if (getAType(axiom) = {"=" || rel}) then
406.     int j = getNumberOfSecondVariable(axiom);
407.     array[j] = true;
408.   endif
409. endfor
410. for (i=0; i<m; i++) do
411.   if (!array[i]) then
412.     return false;
413.   else
414.     return true;
415.   endif
416. endfor

```

4.2 Disjunctive Concepts at Nodes

Now, we allow for the concepts of nodes to contain conjunctions and disjunctions in any order. As from Table 4, *axioms* is the same for all the tests. However, $context_1$ and $context_2$ may contain any number of disjunctions. Some of

them are coming from the concepts of labels, while others may appear from the negated *context*₁ or *context*₂ (e.g., see less/more generality tests). With disjunctive concepts at nodes, formula (1) is a full propositional formula, and hence, no hypothesis can be made on its structure. Thus, its satisfiability must be tested by using a standard SAT decider.

In order to avoid the exponential space explosion, which may arise when converting formula (1) into CNF, we apply a set of structure preserving transformations [23]. The main idea is to replace disjunctions occurring in the original formula with newly introduced variables and to explicitly state that these variables imply the subformulas they substitute. Therefore, the size of the propositional formula in CNF grows linearly with respect to the number of disjunctions in the original formula. Thus, `nodeMatch` should be optimized by replacing all the calls to `convertToCNF` with calls to `optimizedConvertToCNF`.

5 Semantic Matching with Attributes

XML elements may have attributes. Attributes are $\langle \text{attribute} - \text{name}, \text{type} \rangle$ pairs associated with elements. Names for the attributes are usually chosen such that they describe the roles played by the domains in order to ease distinguishing between their different uses. For example, in A1, the attributes *PID* and *Name* are defined on the same domain *string*, but their intended uses are the internal (unique) product identification and representation of the official product's names respectively. There are no strict rules telling us when data should be represented as elements, or as attributes, and obviously there is always more than one way to encode the same data. For example, in A1, *PIDs* are encoded as *strings*, while in A2, *IDs* are encoded as *ints*. However, both attributes serve for the same purpose of the unique product's identification. These observations suggest two possible ways to perform semantic matching with attributes: (i) taking into account datatypes, and (ii) ignoring datatypes.

The semantic matching approach is based on the idea of matching concepts, not their direct physical implementations, such as elements or attributes. If names of attributes and elements are abstract entities, therefore, they allow for building arbitrary concepts out of them. Instead, datatypes, being concrete entities, are limited in this sense. Thus, a plausible way to match attributes using the semantic matching approach is to discard the information about datatypes. In order to support this claim, let us consider both cases in turn.

5.1 Exploiting Datatypes

In order to reason with datatypes we have created a *datatype ontology*, O_D , specified in OWL [26]. It describes the most often used XML schema built-in datatypes and relations between them. The backbone taxonomy of O_D is based on the following rule: *the is-a relationship holds between two datatypes iff their value spaces are related by set inclusion*. Some examples of axioms of O_D are: $\text{float} \sqsubseteq \text{double}$, $\text{int} \perp \text{string}$, $\text{anyURI} \sqsubseteq \text{string}$, and so on. Let us discuss how datatypes are plugged within the four macro steps of the algorithm.

Steps 1,2. Compute concepts of labels and nodes. In order to handle attributes, we extend propositional logics with the quantification construct and datatypes. Thus, we compute concepts of labels and concepts of nodes as formulas in description logics (DL), in particular, using $\mathcal{ALC}(\mathcal{D})$ [22]. For example, $C1_7$, namely, the concept of node describing all the string data instances which are the names of electronic photography products is encoded as $Electronics_1 \sqcap (Photo_1 \sqcup Cameras_1) \sqcap \exists Name_1.string$.

Step 3. Compute relations among concepts of labels. In this step we extend our library of element level matchers by adding a *Datatype* matcher. It takes as input two datatypes, it queries O_D and retrieves a semantic relation between them. For example, from axioms of O_D , the *Datatype* matcher can learn that $float \sqsubseteq double$, and so on.

Step 4. Compute relations among concepts of nodes. In the case of attributes, the node matching problem is translated into a DL formula, which is further checked for its unsatisfiability using sound and complete procedures. Notice that in this case we have to test for modal satisfiability, not propositional satisfiability. The system we use is Racer [15]. From the example in Figure 1, trying to prove that $C2_{10}$ is less general than $C1_9$, requires constructing the following formula:

$$\begin{aligned} & ((Electronics_1 = Electronics_2) \sqcap (Photo_1 = Photo_2) \sqcap \\ & (Cameras_1 = Cameras_2) \sqcap (Price_1 = Price_2) \sqcap (float \sqsubseteq double)) \sqcap \\ & (Electronics_2 \sqcap (Cameras_2 \sqcup Photo_2) \sqcap \exists Price_2.float) \sqcap \neg \\ & (Electronics_1 \sqcap (Photo_1 \sqcup Cameras_1) \sqcap \exists Price_1.double) \end{aligned} \quad (7)$$

It turns out that formula (7) is unsatisfiable. Therefore, $C2_{10}$ is less general than $C1_9$. However, this result is not what the user expects. In fact, both $C1_9$ and $C2_{10}$ describe prices of electronic products, which are photo cameras. The storage format of *prices* in A1 and A2 (i.e., *double* and *float* respectively) is not an issue at this level of detail.

Thus, another semantic solution of taking into account datatypes would be to build abstractions out of the datatypes, e.g., *float*, *double*, *decimal* should be abstracted to type *numeric*, while *token*, *name*, *normalizedString* should be abstracted to type *string*, and so on. However, even such abstractions do not improve the situation, since we may have, for example, an *ID* of type *numeric* in the first schema, and a conceptually equivalent *ID*, but of type *string*, in the second schema. If we continue building such abstractions, we result in having that *numeric* is equivalent to *string* in the sense that they are both datatypes.

The last observation suggests that for the semantic matching approach to be correct, we should assume, that all the datatypes are equivalent between each other. Technically, in order to implement this assumption, we should add corresponding axioms (e.g., $float = double$) to the premises of formula (1). On the one hand, with respect to the case of not considering datatypes (see, Section 5.2), such axioms do not affect the matching result from the quality viewpoint. On

the other hand, datatypes make the matching problem computationally more expensive by requiring to handle the quantification construct.

5.2 Ignoring Datatypes

In this case, information about datatypes is discarded. For example, $\langle Name, string \rangle$ becomes $Name$. Then, the semantic matching algorithm builds concepts of labels out of attribute's names in the same way as it does in the case of element's names, and so on. Finally, it computes mappings using the optimized algorithm of Section 4. A part of the `cNodesMatrix` with relations holding between attributes for the example of Figure 1 is presented in Table 5. Notice that this solution allows us for a mapping's computation not only between the attributes, but also between attributes and elements.

Table 5. Attributes: the matrix of semantic relations holding between concepts of nodes (the matching result)

	$C2_7$	$C2_8$	$C2_9$	$C2_{10}$
$C1_6$	=	idk	idk	idk
$C1_7$	idk	\sqsupseteq	idk	idk
$C1_8$	idk	idk	=	idk
$C1_9$	idk	idk	idk	=

The task of determining mappings typically represents a first step towards the ultimate goal of, for example, data translation, query mediation, data integration, agent communication, and so on. Although information about datatypes will be necessary for accomplishing an ultimate goal, we do not discuss this issue any further since in this paper we concentrate only on the mappings discovery task.

6 Comparative Evaluation

In this section, we present the quality and performance evaluation of the matching system we have implemented, called S-Match. In particular, we validate basic and optimized versions of our system, called $(S\text{-Match}_b)$ and $(S\text{-Match}_o)$ respectively, and evaluate them against three state of the art matchers, namely Cupid [17], COMA [8]¹, and SF [19] as implemented in Rondo [20]. All the systems under consideration are fairly comparable because they are all schema-based. They differ in the specific matching techniques they use and in how they compute mappings.

In our evaluation we have used five pairs of schemas: two artificial examples, a pair of product schemas (our running example, i.e., A1 vs. A2), a pair of

¹ We thank Phil Bernstein, Hong Hai Do, and Erhard Rahm for providing us with Cupid and COMA. In the evaluation we use the version of COMA described in [8]. A newer version of the system COMA++ exists but we do not have it.

Table 6. Some indicators of the complexity of the test cases

	#nodes	max depth	#labels per tree	concepts of nodes
Artificial Example #1	250/500	16/15	250/500	conjunctive
Artificial Example #2	10/10	10/10	30/30	disjunctive
A1 vs. A2	13/14	4/4	14/15	conjunctive & disjunctive
CIDX vs. Excel	34/39	3/3	56/58	conjunctive & disjunctive
Google vs. Looksmart	706/1081	11/16	1048/1715	conjunctive & disjunctive

purchase order schemas (CIDX vs. Excel), and a pair of parts of web directories (Google vs. Looksmart). Table 6 provides some indicators of the complexity of the test cases². As match quality measures we have used the following indicators: *precision*, *recall*, *overall*, *F-measure* (see, [8]). *Precision* varies in the [0,1] range; the higher the value, the smaller is the set of wrong mappings (false positives) which have been computed. *Precision* is a correctness measure. *Recall* varies in the [0,1] range; the higher the value, the smaller is the set of correct mappings (true positives) which have not been found. *Recall* is a completeness measure. *F-measure* varies in the [0,1] range. The version computed here is the harmonic mean of *precision* and *recall*. It is a global measure of the matching quality, growing with it. *Overall* is an estimate of the post-match efforts needed for adding false negatives and removing false positives. *Overall* varies in the [-1,1] range; the higher it is, the less post-match efforts are needed. As a performance measure we have used *time*. It estimates how fast systems are when producing mappings fully automatically.

To provide a ground for evaluating the quality of match results, initially, the schemas have been manually matched to produce expert mappings. Then, the results computed by systems have been compared with expert mappings. There are three further observations that ensure a fair comparative study. The first observation is that Cupid, COMA, and Rondo can discover only the mappings which express similarity between schema elements. Instead, S-Match, among the others, discovers the disjointness relation which can be interpreted as strong dissimilarity in terms of the other systems under consideration. Therefore, we did not take into account the disjointness relations (e.g., $\langle ID_{4,4}, C1_4, C2_4, \perp \rangle$) when specifying the expert mappings. The second observation is that, since S-Match returns a matrix of relations, while all the other systems return a list of the best mappings, we used some filtering rules. More precisely we have the following two rules: (i) discard all the mappings where the relation is *idk*; (ii) return always the *core* relations, and discard relations whose existence is implied by the core relations. For example, $\langle ID_{3,3}, C1_3, C2_3, = \rangle$ should be returned, while $\langle ID_{3,5}, C1_3, C2_5, \sqsupseteq \rangle$ should be discarded. Finally, whether S-Match returns the equivalence or subsumption relations does not affect the quality indicators. What only matters is the presence of the mappings standing for those relations.

In our experiments each test has two degrees of freedom: *directionality* and *use of oracles*. By directionality we mean here the direction in which mappings

² Source files, description of the test cases, and expert mappings can be found at <http://www.dit.unitn.it/~accord/>, experiments section.

have been computed: from the first schema to the second one (forward direction), or vice versa (backward direction). For lack of space we report only the best results obtained with respect to directionality, and use of oracles allowed. We were not able to plug a thesaurus in Rondo, since the version we have is standalone, and it does not support the use of external thesauri. Thesauri of S-Match, Cupid, and COMA were expanded with terms necessary for a fair competition (e.g., expanding *uom* into *unitOfMeasure*, a complete list is available at the URL in footnote 2).

All the tests have been performed on a P4-1700, 512 MB of RAM, Windows XP, with no applications running but a single matching system. Notice, that the systems were limited to allocate no more than 512 MB of main memory. Also, all the tuning parameters (e.g., thresholds, strategies) of the systems were taken by default (e.g., for COMA we used NamePath and Leaves matchers combined in the Average strategy) for all the tests.

6.1 Test Cases

Let us discuss artificially designed problems in order to evaluate the performance of S-Match_o in *ideal* conditions, namely when we have only conjunctive or disjunctive concepts of nodes. Since examples are artificial and our optimizations address only efficiency, not quality, we analyze here only the performance *time* of the systems, see, Figure 2 (Artificial Examples).

On the example with conjunctive concepts at nodes (Artificial Example #1), COMA performs 4 times slower and 15 times slower than S-Match_b and S-Match_o, respectively. S-Match_o runs around 29% faster than Rondo. Instead, Cupid runs out of memory.

On the example with disjunctive concepts at nodes (Artificial Example #2), S-Match_o works around 4 orders of magnitude faster than S-Match_b, around 5 times faster than COMA, 1.6 times faster than Cupid, and as fast as Rondo. The significant improvement of our optimized algorithm can be explained by considering that S-Match_b does not control the exponential space explosion on such matching problems. In fact, the biggest formula in this case consists of about 118000 clauses. The optimization introduced in the Section 4.2 reduces this number to approximately 20-30 clauses.

We have then considered 3 matching problems, also involving real-world examples. Let us first discuss matching results from our running example, see, Figure 2 (Product schemas: A1 vs. A2). There, S-Match outperforms the other systems in terms of quality indicators. Since all the labels at nodes in the given test case were correctly encoded into propositional formulas, all the quality measures of S-Match reach their highest values. In fact, as discussed before, the propositional SAT solver is correct and complete. This means that once the element level matchers have found all and only the mappings, S-Match will return all of them and only the correct ones. Also, S-Match_o works more than 5 times faster than COMA, 1.5 times faster than Cupid, and as fast as Rondo.

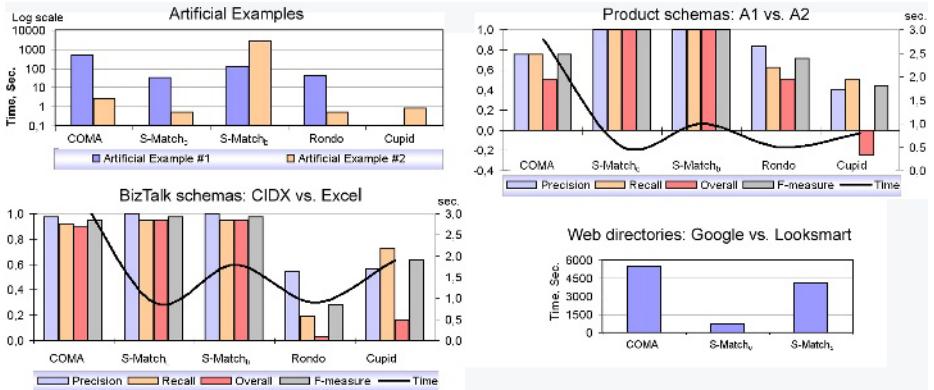


Fig. 2. Evaluation Results

For a pair of BizTalk schemas: CIDX vs. Excel, S-Match performs as good as COMA and outperforms the other systems in terms of quality indicators. Also, S-Match_o works more than 4 times faster than COMA, more than 2 times faster than Cupid, and as fast as Rondo.

For the biggest matching problem (Web Directories: Google vs. Looksmart), which contains hundreds and thousands of nodes, unfortunately, we did not have enough human resources to create expert mappings for this test case (we are still working on establishing them), and thus, for the moment we have evaluated only the performance *time*. S-Match_o performs about 9 times faster than COMA, and about 7 times faster than S-Match_v. Rondo and Cupid run out of memory, therefore we do not report any results for them.

6.2 Evaluation Summary

Quality measures. Since most matching systems return similarity coefficients, rather than semantic relations, our qualitative analysis was based on the measures developed for those systems. Therefore, we had to omit information about the type of relations S-Match returns, and focus only on the number of present/absent mappings. We totally discarded from our considerations the disjointness relation, however, its value should not be underestimated, because this relation reduces the search space. For the example of Figure 1, if Cupid would support the analysis of dissimilarity between schema elements, it could possibly recognize that $C1_4$ is disjoint (dissimilar) with $C2_4$, and then avoid false positives such as determining that $C1_{10}$ is similar to $C2_{11}$, and so on.

Pre-match efforts. Typically, these efforts include creating a precompiled thesaurus with relations among common and domain specific terms. On the one side, such a thesaurus can be further reused, since many schemas to be matched are similar to already matched schemas, especially if they are describing the same application domain. On the other side, for the first schemas to be matched

from a novel domain, creation of such a thesaurus requires time. With this respect, exploiting an external resource of common and domain knowledge (e.g., WordNet) can significantly reduce the pre-match efforts. In the example of Figure 1, in order for Cupid to determine $C1_7$ as an appropriate match for $C2_8$, we have to add an entry $<\text{Hyp key}=\text{"brand:name"}> 0.7 </\text{Hyp}>$ to its thesaurus, while S-Match obtains the knowledge of the hyponymy relation in the above case automatically from WordNet.

Performance measures. *Time* is a very important indicator, because when matching industrial-size schemas (e.g., with hundreds and thousands of nodes, which is quite typical for e-business applications), it shows scalability properties of the matchers and their potential to become an industrial-strength systems. It is also important in web applications, where some weak form of real time performance is required (to avoid having a user waiting too long for the system respond).

7 Conclusions

We have presented a new semantic schema matching algorithm and its optimizations. Our solution builds on the top of the past approaches at the element level and introduces a novel (with respect to schema matching) techniques, namely model-based techniques, at the structure level. We conducted a comparative evaluation of our approach implemented in the S-Match system against three state of the art systems. The results empirically prove the strengths of our approach.

Future work includes development of the *iterative* and *interactive* semantic matching system. It will improve the quality of the mappings by iterating and by focusing user's attention on the critical points where his/her input is maximally useful. S-Match works in a top-down manner, and hence, mismatches among the top level elements of schemas can imply further mismatches between their descendants. Therefore, next steps include development of a *robust* semantic matching algorithm. Finally, we are going to develop a *testing methodology* which is able to estimate quality of the mappings between schemas with hundreds and thousands of nodes. Initial steps have already been done, see for details [1]. Here, the key issue is that in these cases, specifying expert mappings manually is (often) neither desirable nor feasible task. Comparison of matching algorithms on real-world schemas from different application domains will also be performed more *extensively*.

References

1. P. Avesani, F. Giunchiglia, and M. Yatskevich. A large scale taxonomy mapping evaluation. In *Proceedings of ISWC*, 2005.
2. S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, pages 54–59, 1999.
3. P. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-strength schema matching. *SIGMOD Record*, 33(4):38 – 43, 2004.

4. D. Le Berre. A satisfiability library for Java. <http://www.sat4j.org/>.
5. P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: A new approach and an application. In *Proceedings of ISWC*, pages 130–145, 2003.
6. M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, (5(7)):394–397, 1962.
7. R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering complex semantic matches between database schemas. In *Proceedings of SIGMOD*, pages 383–394, 2004.
8. H. H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proceedings of VLDB*, pages 610–621, 2002.
9. J. Euzenat and P. Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proceedings of ECAI*, pages 333–337, 2004.
10. A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB Journal*, (14(1)), 2005.
11. F. Giunchiglia and P. Shvaiko. Semantic matching. *KER Journal*, (18(3)), 2003.
12. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *Proceedings of ESWS*, pages 61–75, 2004.
13. F. Giunchiglia, M. Yatskevich, and E. Giunchiglia. Efficient semantic matching. In *Proceedings of ESWC*, pages 272–289, 2005.
14. N. Guarino. The role of ontologies for the Semantic Web (and beyond). Technical report, Laboratory for Applied Ontology, ISTC-CNR, 2004.
15. V. Haarslev, R. Moller, and M. Wessel. RACER: Semantic middleware for industrial projects based on RDF/OWL, a W3C Standard. <http://www.sts.tu-harburg.de/~r.f.moeller/racer/>.
16. J. Kang and J. F. Naughton. On schema matching with opaque column names and data values. In *Proceedings of SIGMOD*, pages 205–216, 2003.
17. J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of VLDB*, pages 49–58, 2001.
18. B. Magnini, L. Serafini, and M. Speranza. Making explicit the semantics hidden in schema models. In *Proceedings of workshop on HLTSWWS at ISWC*, 2003.
19. S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. In *Proceedings of ICDE*, pages 117–128, 2002.
20. S. Melnik, E. Rahm, and P. Bernstein. Rondo: A programming platform for generic model management. In *Proceedings of SIGMOD*, pages 193–204, 2003.
21. A. G. Miller. WordNet: A lexical database for English. *Communications of the ACM*, (38(11)):39–41, 1995.
22. J. Z. Pan. *Description Logics: reasoning support for the Semantic Web*. PhD thesis, School of Computer Science, The University of Manchester, 2004.
23. D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, (2):293–304, 1986.
24. E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, (10(4)):334–350, 2001.
25. P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, IV, 2005.
26. M. K. Smith, C. Welty, and D. L. McGuinness. OWL web ontology language guide. Technical report, World Wide Web Consortium (W3C), <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>, February 10 2004.

Unified Semantics for Event Correlation over Time and Space in Hybrid Network Environments

Eiko Yoneki and Jean Bacon

University of Cambridge Computer Laboratory
Cambridge CB3 0FD, United Kingdom
{Eiko.Yoneki, Jean.Bacon}@cl.cam.ac.uk

Abstract. The recent evolution of ubiquitous computing has brought with it a dramatic increase of event monitoring capabilities by wireless devices and sensors. Such systems require new, more sophisticated, event correlation over time and space. This new paradigm implies composition of events in heterogeneous network environments, where network and resource conditions vary. Event Correlation will be a multi-step operation from event sources to final subscribers, combining information collected by wireless devices into higher level information or knowledge. Most extant approaches to define event correlation lack a formal mechanism for establishing complex temporal and spatial relationships among correlated events. Here, we will focus on two subjects. First, we define generic composite event semantics, which extend traditional event composition with data aggregation in wireless sensor networks (WSNs). This work bridges data aggregation in WSNs with event correlation services over distributed systems. Secondly, we introduce interval-based semantics for event detection, defining precisely complex timing constraints among correlated event instances.

1 Introduction

An event correlation service is important for constructing reactive distributed applications. It occurs as a part of applications, event notification services and workflow coordinators. In event-based middleware systems, an event correlation service allows consumers to subscribe to patterns of events (composite events). This provides an additional dimension of data management, and improvement of scalability and performance in distributed systems. Particularly in wireless networks, providing event correlation as a middleware service helps to simplify the application logic and reduce its complexity.

The recent evolution of ubiquitous computing has brought with it a significant increase of event monitoring capabilities by wireless devices and sensors. Such systems require new, more sophisticated, event correlation over time and space. Typical Wireless Sensor Networks (WSNs) communicate directly with a centralized controller or a satellite. Thus, communication between a sensor and a controller is based on a single-hop model. On the other hand, a collection of autonomous nodes or terminals may communicate with each other by forming a multi-hop radio network and maintaining connectivity in a decentralized manner, thus creating an ad hoc network. Moreover, the integration of a smart WSN with a large network such as the Internet, increases its coverage and potential application domain. In WSNs, a sink node is a sensor node with

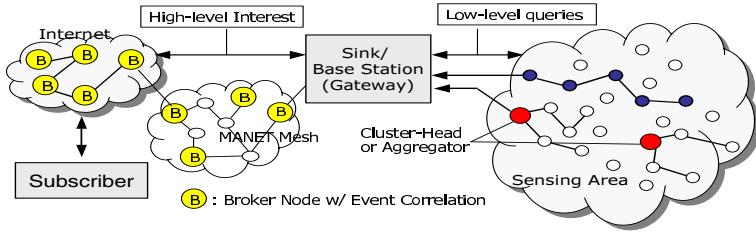


Fig. 1. Bridging WSNs to the Internet

gateway functions to link to external networks such as the Internet. Sensed information is normally distributed via a sink node. A sink node may also be an Internet backbone node, or the gateway node to an intermediate ad hoc network, which may deliver the sensor data to the Internet. Fig.1 depicts WSNs connecting to the Internet, where an ad hoc network (the MANET Mesh in Fig.1) could form an opportunistic network such as [3].

This new platform enables the seamless use of the various resources in physically interacting environments. A consensus is emerging that the most appropriate system architecture to support such platforms is service management, with communication based on the publish/subscribe paradigm. For example, a publisher broker node can act as a gateway from a WSN, performing data aggregation and distributing filtered data to other networks based on contents. Event broker nodes that offer data aggregation services can coordinate data flow efficiently. Especially when event-based communication is implemented via a peer-to-peer (P2P) overlay network, the construction of event broker grids will extend the seamless messaging capability over scalable heterogeneous network environments. Event Correlation will be a multi-step operation from event sources to the final subscribers, combining information collected by wireless devices into higher level information or knowledge. Mobile devices can be deployed in remote locations without a network infrastructure. They will have an important role in collecting sensor data over ad hoc networks and conveying it to Internet backbone nodes. These mobile devices are resource constrained, and an implementable event detection mechanism is required. The semantics of operators for composite events is not defined in a uniform manner in existing middleware and applications leading to a number of problems. Event consumption rules are mostly done as part of an implementation, without a clear semantic definition. Most extant approaches to define event correlation lack a formal mechanism to define complex temporal and spatial relationships among correlated events. Thus, a unified semantics has to be defined to resolve this ambiguity for heterogeneous network environments.

Temporal ordering in real-time is a critical aspect of event correlation in wireless ad hoc network environments. The context of real-time in this paper relates to real-world event occurrences. Neither logical time nor classical physical clock synchronization algorithms may be applicable. Temporal ordering is required for motion detection. In order to determine the direction of movement of a real-world entity, temporal ordering of events originating from different devices has to be established. Events can be triggered by physical phenomena, such as glaciers and earthquakes, and the order of occurrence of sensed data is again important. In general, recent developments in Internet business

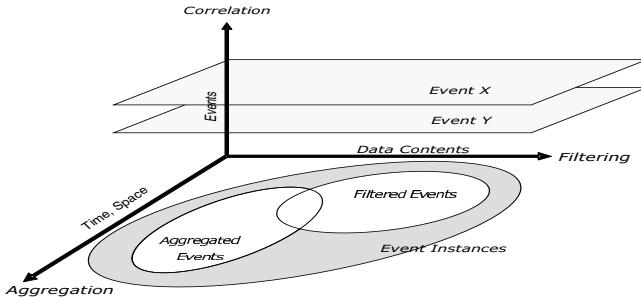


Fig. 2. Aggregation, Filtering and Correlation

demand precision time for processing data. For example, orders to buy and sell on the stock market and in an auction must be timed on a global scale, and a trigger event may be initiated by a wireless devices.

Event Aggregation, Filtering and Correlation: Some event-based middlewares offer content-based filtering and provide flexible query languages. These allow subscribers to select events of interest, based on the values of their contents. A query can apply to different event types but the aim is to select individual events. On the other hand, event correlation addresses the relationship among, or pattern of, instances of different event types. WSNs have led to new issues to be addressed in event correlation. Traditional networking research approached data aggregation as an application-specific technique that can be used to reduce network traffic. In WSNs the requirement is to summarize current sensor values in some or all of a sensor network.

TinyDB [19] is an inquiry processing system for sensor networks and takes a data centric approach. Each node keeps the data and executes retrieval and aggregation (in-network aggregation), with on-demand based operation to deliver the data to external applications. TinyLIME [5] enhances LIME (Linda In Mobile Environments) to operate on TinyOS. In TinyLIME, LIME is maintained on each sensor node together with a partition of a tuple space. A coordinated tuple space is created across the nodes, connecting with the base station in one hop. TinyLIME works as middleware by offering this abstracted interface to the application. It does not currently provide any data aggregation function, only a data filtering function based on Linda/LIME at the base station node. On the other hand, TinyDB supports data aggregation via SQL query, but redundancy/duplication handling is not clear from available documents. Fig.2 shows the relationships between aggregation, filtering and correlation. Middleware research for WSNs has been active recently, but most research focuses on in-network operation for specific applications. In this paper, we take a global view of event correlation over entire distributed systems, focusing on correlation semantics.

Unified Semantics for Event Correlation: We define a unified semantics, combining traditional event composition and data aggregation in wireless sensor networks. In-network aggregation in WSNs is not a concern of this paper. For the event detection semantics, we introduce a parameterized algebra. Parameters include time, selection, consumption, and subset rules. This approach defines unambiguous semantics of event

detection and supports resource constrained environments. We introduce interval-based semantics for event detection defining precisely complex timing constraints among correlated event instances. In resource constrained network environments, the event algebra must be restricted so that only a subset of all possible occurrences of complex events will be detected, and this can be achieved by applying appropriate parameters. This paper continues as follows: Section 2 describes key aspects of event composition semantics with existing systems as examples. Section 3 describes the event model, Section 4 defines composite event semantics and section 5 discusses temporal ordering. The formal definition and proof of the event algebra is out of the scope of this paper. In Section 6 we describe related work, and Section 7 contains conclusions and directions for future research.

2 Event Correlation in Middleware

In this section, we show a comparative study of existing event correlation mechanisms in middleware. Event correlation may be deployed as a part of applications, as event notification services, or as part of a middleware framework. Definition and detection of composite events vary, especially over distributed environments. Equally, named event composition operators do not necessarily have the same semantics, while similar semantics might be expressed using different operators. Moreover, the exact semantic description of these operators is rarely explained. Thus, we define the following schema to classify existing operators: conjunction, disjunction, sequence, concurrency, negation, iteration, and selection. Considering the analyzed systems, it becomes clear that to simply consider the operators is not sufficient in order to convey the full semantic meaning. Each system offers parameters, which further define/change the operators' semantics. The problem is that the majority of the systems reflect parameters within the implementations. Parameters for consumption mode and duplicate handling are rarely explicitly described. Table 1 and 2 shows a comparative study of event composition semantics in ten existing systems and our proposal ECCO. The tables give an overview of event-based composition languages typically supported in event-based systems, and provide an analysis of the languages from a unified viewpoint. Note that the list of analyzed systems cannot be exhaustive, but considers a representative set of selected composition semantics. None of the listed systems includes wireless network environments except ECCO. Most event notification systems still only support primitive events, and their focus is on efficient filter algorithms. Brief explanations of the characteristics of each system are given below (see also Section 6, related work).

Table 1 shows composition operators, while Table 2 emphasizes temporal order related parameters including consumption modes. Concepts of conjunction, disjunction, sequence, concurrency, negation, and iteration operators are based on an event algebra. Selection defines the event instance selection, by which events qualify for a composite event, and how duplicate events are handled. Conjunction and disjunction are supported in most systems, some of which implement the sequence operators (requiring ordering), fewer support negation. Selection and concurrency are rarely supported. Concurrency is difficult to determine for distributed systems. Time operators are not always supported, requiring a time handling strategy for distributed systems. Consumption mode

Table 1. Event Composition Semantics - Composition Operators

	Operators						
	Conjunction	Disjunction	Sequence	Conc.	Negation	Iteration	Selection
ECCO	$A + B$	$A B$	$A; B$	$A B$	$\neg A$	A^*	A^N
Opera	$A B$	$A B$	$A; B$ or AB	-	$\neg A$	A^*	-
CEA	$A \& B$	$A B$	$A; B$	-	$\neg A$	-	-
Schwiderski	A, B	$A B$	$A; B$	$A B$	$NOTA$	$A^* \text{ or } A^+$	-
A-mediAS	$A \& B$	$A B$	$A; B$	-	$\neg A$	-	$A^{[i]}$
Ready	$A \&& B$	$A B$	$A; B$	-	$notA$	-	-
Eve	$CON(A, B)$	$DEX(A, B)$	$SEQ(A, B)$	$CCR(A, B)$	$NEG(A, B)$	$REP(A, n)$	-
GEM	$A \& B$	$A B$	$A; B$	-	$\neg A$	-	-
Snoop	A, B	$A \vee B$	$A; B$	-	-	A^*	-
Rebeca	$A \wedge B$	$A \vee B$			$\neg A$	A^*	-
SAMOS	A, B	$A B$	$A; B$	-	$NOTA$	$TIMES(n, A)$	$A^* / last(A)$

and temporal conditions are rarely made explicit. If they are explicit, several options are supported, otherwise they are hard-coded in the system and difficult to determine. The listed systems are notification services, event composition languages, and workflow coordinators in which common characteristics are fairly complete semantics of event composition.

ECCO: is our ongoing project to create an event brokering architecture for a distributed adaptive mobile environment [27]. Event correlation is part of event brokers, and grids of brokers are deployed over mixed network environments. The ECCO prototype is implemented with a MANET protocol as content-based publish/subscribe. This paper focuses on event correlation over event broker grids, which requires correlation of events from various network environments.

Opera: a framework for event composition in a large scale distributed system [23] aiming at reduction of event traffic by distributed composite event detectors. The language of composite events is based on FSA.

CEA: our early work on the Cambridge Event Architecture (CEA) extended the then-predominant, object-oriented middleware (CORBA and Java) with a *publish, register and notify* paradigm [11]. COBEA [21] is an event-based architecture used for the management of networks using CEA based composite event operators.

Schwiderski: enhanced the distributed event ordering and composite event detection by introducing 2g-precedence-based sequence and concurrency operators [26].

A-mediAS: an integrating event notification service that is adaptable to different applications specifically on handling composite events and event filtering methods [12].

Ready: an event notification service from AT&T Research similar to Siena. Ready supports composite events and its grouping functionality can be shared among clients [10].

EVE: combines characteristics of active databases and event-based architectures to execute event driven workflows [8].

GEM: GEM [20] is an interpreted generalized event monitoring rule based language.

Snoop: a model-independent event specification language [4], supporting parameter contexts. It supports temporal, explicit and composite events for active databases.

Rebeca: an event-based electronic commerce architecture focusing on event filtering in a distributed environment [22]. Temporal delays in event composition have been addressed in [16].

SAMOS: The SAMOS (Swiss Active Mechanism-based Object-oriented database system) [7] project addresses the specification of active behavior and its internal processing supporting Event-Condition-Action (ECA) rules. The detection of composite events is implemented based on colored Petri-Nets.

Table 2. Event Composition Semantics - Time-related Parameters

	Time Operator		Timestamp	Composition	Temp. Cond.	Consumption			Subset	Detection
	Period	Life				Unrest.	Recent	Chron.		
ECCO	$(A; B)_T, (A + B)_T$	$((A)_T$	P, PI, I	I	×	×	×	×	×	Algorithm
Opera	$(A, B)_T$	-	PI	P	-	-	-	-	-	T-FSA
CEA	-	-	P	P	-	×	-	-	×	FSA
Schwiderski	-	-	P	P	-	-	-	-	-	Rule
A-mediAS	$(A, B)_T, (A; B)_T, -A_T$	-	P	P	-	×	×	×	×	T-FSA
Ready	-	-	P	P	-	-	-	-	×	-
Eve	-	-	P	P	-	-	-	×	×	Graph
GEM	(A+timeperiod)	-	P	P	×	-	×	-	×	Rule
Snoop	$(A, [timestring] : param, B)$	-	P	P	-	-	×	×	×	Graph
Rebeca	slide window	-	PI	P	-	-	×	×	-	Rule
SAMOS	-	-	-	-	-	-	-	×	×	Petri net

Time Operator: Period (between event A and B) Life (valid time for event A)

Timestamp: P (Point-based), PI (Point represented in Interval-based format), and I (Interval-based).

Composition: Event composition semantics. P (Point-based), and I (Interval-based).

Temporal Condition: Temporal conditions such as *A before B*, *A meets B*, etc. for event composition.

Consumption: Event consumption (Unrestricted, Recent, and Chronicle).

Subset: Parameters for selection or subset on duplication handling.

Detection: Implementation methods.

3 Event Model

In this section, we define our event model including timestamps for composite events. A primitive event is the occurrence of a state transition at a certain point in time. Each occurrence of an event is called an event instance. The primitive event set contains all primitive events within the system. The event set consists of the set of primitive events and the set of composite events. Each event has a timestamp associated with the occurrence time. There is uncertainty associated with the values of timestamps in implemented systems. Composite events are defined by composing primitive or composite events with a set of operators. Only composite events are defined to have duration.

Timestamps: A timestamp is a mandatory attribute of an event defined within a time system, while the event occurrence time is a real-time defined by the occurrence of the event. Thus, the timestamp is an approximation of the event occurrence time. Most

point-based timestamps consist of a single value indicating the occurrence time. In [16], the time when an event is detected is given as an interval-based timestamp, which captures clock uncertainty and network delay with two values: the low and high end of the interval. Although an interval format is used, it represents a single point (point-interval-based timestamp). In addition, we define a (composite) event with duration and give a new interval-based timestamp to a composite event based on interval semantics (see Section 4.3). For a primitive event, either a point-based or point-interval-based timestamp is used in our system. A point-interval-based timestamp is an accurate representation, and it is distinct from interval-based timestamps representing the duration of events. Fig.3 depicts an interval-based timestamp for composite events.

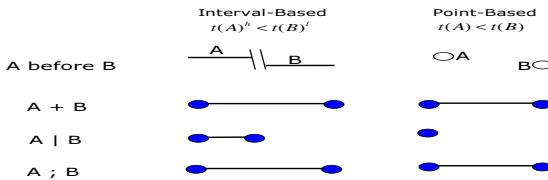


Fig. 3. Timestamp of Composite Event (A before B)

Spacestamp: We introduce spacestamp, as an optional attribute of an event, indicating certain location, relative location, grouping and so forth (e.g., position information (x,y,z), global node id). The Global Positioning System (GPS) [13] provides each node with its location information (latitude, longitude and elevation) with a high degree of accuracy. This information can be used for ordering events within the given space.

3.1 Duration

Reference [1] argues that all events have duration and considers intervals to be the basic timing concept. A set of 13 relations between intervals is defined, and rules governing the composition of such relations controls temporal reasoning. On the other hand, most event systems listed in Section 2 consider events as instantaneous, that is, the time associated with the event is an instant rather than an interval. A durative event can be seen as capturing the uncertainty over the time of occurrence and the time of detection of an event rather than modelling an event that persists over time. In this sense, durative events are akin to the point-interval-based-timestamps described above. The durative event model considers that instantaneous events are durative events with minimum duration, thus reconciling the models. Our model is based on primitive events that represent instantaneous changes of system state, with uncertainty over their measurement. We would regard an “event” that persists over time as akin to a state, with an event at the start and one at the end of the time period. This could also be defined as a composite event. Composite events are built up from events occurring at different times, therefore the associated real-time is usually that of the last of its contributory primitive

events. This is natural in a context where the prime focus is on event detection, since typically a composite event will be detected at the time that its last contributory event is detected. However, this does lead to logical difficulties in the case of some composite events. Determination of the duration of composite events requires the semantics of composition and time system information such as a point-based or an interval-based time model. Fig.4 shows an example of a sequence operation on the interval-based and point-based timestamps. Complex timing constraints among correlated event instances are precisely defined (see Appendix).

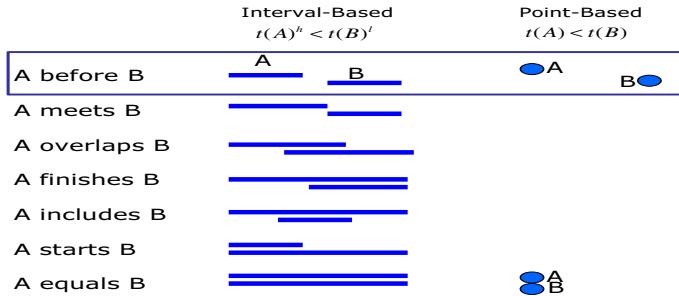


Fig. 4. Interval and Point based Timestamps

3.2 Duplication

It is important to distinguish between multiple instances of a given event type, which may be primitive or composite, and duplicates of a given event instance. The expressiveness of some event specification languages has been limited by not distinguishing between event types and instances of those types. [17] attempts to define conditions and constraints on attributes of events in correlation rules rather than defining operators on event instances. Especially in sensor networks, in order to avoid loss of events by communication instability, duplicates of events may be produced to increase reliability. Duplicates have to be handled differently depending on the application, and contexts within applications. For example, in object tracking, the most recent reading from a sensor is valid, and events prior to that will be obsolete, except for the historical record. On the other hand, in a transaction event in which a customer cancels an order, a duplicate event should be ignored because a transaction is being repeated. Thus, the semantics of event composition have to address handling of duplicates. [18] take the approach of defining constraints on attributes of events and detect occurrences of events, before correlation conditions are evaluated. We propose duplicate handling in two ways: adding a selection operator as an event composition operator and adding subset rules as parameters (obtaining this efficiently without loss of meaningful data) (see Section 4.4).

4 Event Correlation Semantics

We define composite events by expressions built from primitive and composite events and algebraic operators. The operators of the event algebra are defined informally in

this section. We also support parameters, which help to define unambiguous semantics of event detection and support resource constrained environments. In wireless ad hoc networks, the event algebra must be restricted so that only a subset of all possible occurrences of complex events will be detected. We provide basic operators that have the potential of expressing the required semantics and are capable of restricting expressions. Also, an interval-semantics supports more sensitive interval relations among events in environments where real-time concerns are more critical, such as wireless networks or multi-media systems. Our recent work [23] defined composite event operators that are tightly based on FSA without considering time-related issues in depth. The temporal operators introduced in [1] are not uniformly defined in many applications, and we define precise timing constraints (see Appendix and [28]). The temporal operators help resource constrained mobile devices to have unambiguous semantics.

4.1 Composite Event Operators

The event operators are defined informally as follows:

- **Conjunction A + B:** Event A and B occur in any order. $(A + B)_T$ with a temporal parameter T indicating the maximal length of the interval between the occurrences of A and B. Note that $(A + B)_{\infty}$ or $(A + B)$ refers without restrictions.
- **Disjunction A | B:** Event A or B occurs.
- **Concatenation A B:** Event A occurs before event B where timestamp constraints are *A meets B, A overlaps B, A finishes B, A includes B, and A starts B*.
- **Sequence A ; B:** Event A occurs before B where timestamp constraints are *A before B, and A meets B*. $(A;B)_0$ is a special case belonging to *A meets B*.
 - E.g. $(A;NULL;B)$: denotes there is no occurrence of any event between event A and B.
 - E.g. $(A ; B)_T$: means that an interval T between event A and B.
 - E.g. $(A;B_0)$: denotes that there is event A and event B occur contiguously.
- **Concurrency A||B:** Event A and B occur in parallel.
- **Iteration A^{*}:** Any number of event A occurrences.
- **Negation -A_T:** No event A occurs for an interval T.
 - E.g. $(A - B)$: denotes no B occurs during A's occurrence.
 - E.g. $(A - B)_T$: denotes no B occurs after starting A's occurrence within an interval T.
 - E.g. $(A; B) - C$: denotes that event A is followed by B and there is no C in the duration of $(A;B)$.
- **Selection A^N:** The selection A^N defines the occurrence defined by N.
 - E.g. A_T^{AVG} : denotes taking the average during an interval T.
 - E.g. A_T^{LAST} : denotes taking the most recent instance during an interval T.
- **Spatial Restriction A_S:** Event A occurs if it is a spatial restriction defined in S, that can be defined as a specific location or a group identifier etc.
 - E.g. A_{CB03FD} : The area code CB03FD identifies the zone around Computer Laboratory in Cambridge. Event A is valid only when spatial condition is satisfied.
- **Temporal Restriction A_T:** Event A occurs within T.
 - E.g. $(A; B)_T$ or $(A; B_T)$: B occurs within an interval T after A.
 - E.g. B_T : B is valid for an interval T.

The following examples illustrate the use of the operators to describe composite events.

Example 1: The temperature of rooms with windows facing south is measured every minute and transmitted to a computer placed on the corridor. T denotes a temperature event and T_{30}^{AVG} denotes a composite event of an average of the temperature during 30 minutes. $(T_{room1} + T_{room7})_{30}^{AVG}$ denotes to take an average of room 1 and 7.

Example 2: Two sensing receivers are placed before and after the stop sign on the street. When a car passes, they generate events to the local computer. Suppose the event received before the stop sign is B and after the stop sign is A for a given car. $(B; A)_2$ denotes A occurs 2 seconds after B and it indicates a car did not make a full stop at the stop sign. On the other hand, $((B; A)_{60})^*$ indicates cars may not be flowing on the street, which indicates potential traffic congestion.

Example 3: At a highway entrance, a sensor detects movement of a passing car as event E . The number of cars entering the highway $(E_{10}^{SUM})_{HWY1Ent7}$ can be calculated at a computer locally, which can be used to detect traffic congestion on the highway (e.g., a congestion event $C = ((E_{10}^{SUM})_{HWY1Ent7})^{LESS12}$).

4.2 Temporal Conditions

Defining temporal conditions for the semantics of composite events could be tricky, especially when timing constraints are important such as for processing transactions. This may cause an incorrect interpretation according to the intuition of the user. Fig.5

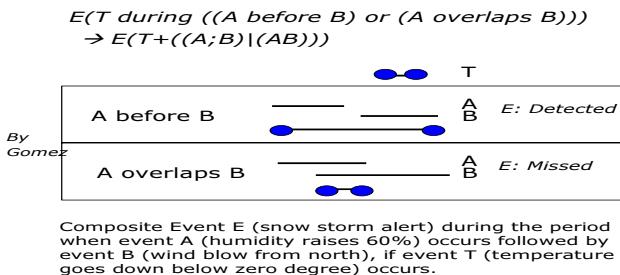


Fig. 5. Semantic Ambiguity

depicts a composite event E (*snow storm alert*): during the period when primitive event A (*humidity raises 60%*) occurs followed by primitive event B (*wind blows from north*), if primitive event T (*temperature goes down below zero degree*) occurs. Two situations are shown. If we follow the interpretation of temporal conditions described by Gomez et al. in [9], in the first situation, event E is detected and in the second situation, event E is missed. In [9], *overlaps* and *during* only comprises the period when two events are simultaneously occurring, while every other operator takes the period over both event

occurrences. This inconsistency may cause a problem. In both occasions, the natural interpretation of event E is the same. With our definition, both examples will yield consistent results.

4.3 Interval Semantics

In most event algebras, each event occurrence, including composite events, is associated with a single time point. This may result in unintended semantics for some operator combinations, for example nested sequence operators. In Fig.6, time flows from left to right, and each row shows the occurrence of a primitive event. When single point detection is used, an instance of event $B;(A;C)$ is detected if A occurs first, followed by B and C. The reason is that these occurrences cause a detection of $A;C$, which is associated with the occurrence of $B;(A;C)$. With interval semantics, the sequence $A;B$ can be defined to occur only if the intervals of A and B are non-overlapping. No occurrence of $B;(A;C)$ would be detected.

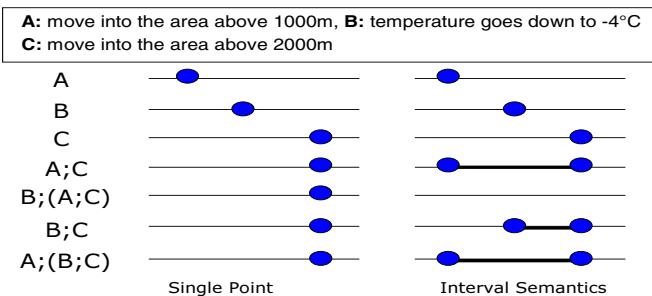
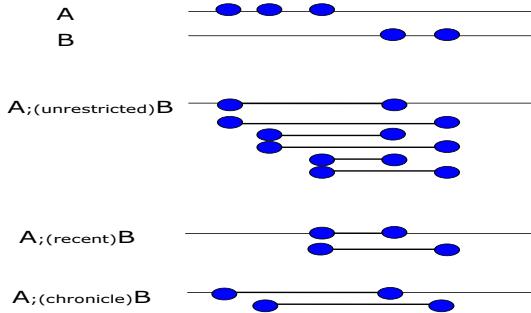


Fig. 6. Point and Interval

4.4 Event Context

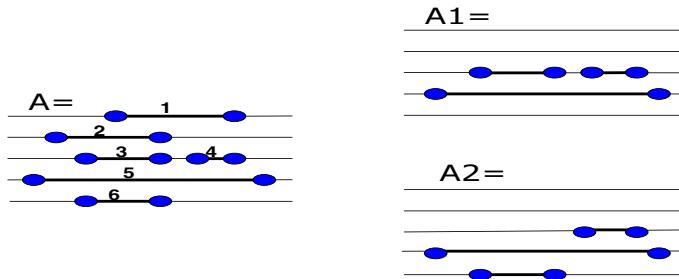
Adding the policy defining the constraints provides a way to modify the operator semantics. This parameter-dependent algebra can accommodate different policies on event consumption. First, each operator is given a principle definition of the constraints on the participating occurrences of events that characterize the operator. Then a number of event contexts are defined that act as modifiers to the simple operator semantics. These contexts specify constraints on how occurrences may be selected. As a result, each combination of an operator and a context can be seen as a separate operator with a specific meaning. This helps to deal with resource constrained environments, e.g., keeping the most recent instance for future use.

Consumption Policy: For event consumption policy, three contexts *unrestricted*, *recent* and *chronicle*, can be defined. Snoop [4] uses these contexts, but it is not capable of applying an individual context to different event operators. The parameter-dependent algebra clarifies the situations. The following gives an informal definition for detecting $A;B$. Fig.7 shows the effect of these contexts on the sequence operator.

**Fig. 7.** Event Consumption Policy

- Unrestricted: All instances of A and B are valid.
- Recent: If an instance of B can be combined with several instances of A to form instances of A;B, the only recent instance of A is valid.
- Chronicle: If an instance of B can be combined with several instances of A to form instances of A;B, only the oldest instance of A is valid. Also, this instance is never valid in the future.

Subset Policy: defines the subset of events to detect. Ideally the *Subset Policy* should interfere as little as possible with unrestricted semantics. None of the removed instances should have a crucial impact on the detection of an enclosing detection. At the same time, operations such as conjunction and sequence must be able to identify non-valid instances early, before the end time of the instance is reached. The main task of the *Subset Policy* is to make an effective algebra, feasible to implement in resource-constrained environments.

**Fig. 8.** Subset Policy

The basis of the *Subset Policy* is that the restricted event stream should be a subset that does not contain multiple instances with the same end time. Informally, from the instances with the same end time, the *Subset Policy* keeps exactly one, that with the maximal start time. Fig. 8 shows the result of applying the *Subset Policy* to an event

stream A. From the three instances of A (2,3,6) with the same end time, 2 must be removed, together with either 3 or 6. For the two instances (1,4) that share the same end time, the one with the earlier time must be removed. 5 does not share the end time with others. The choice of which of (3,6) to remove results in two valid restrictions of the event stream A, named A1 and A2.

Precision Policy: defines the required precision of the events to be detected. The dynamic spatial-temporal data from sensor networks is generated at a rapid rate and all the generated data may not arrive at the event correlation node over the networks due to the lossy/faulty nature of the sensor network. Various techniques, including compression and model adaptation have provided certain levels of guarantees [15]. On the other hand, if some imprecision of the collected data could be tolerated by the application, defining the precision available is important.

For example, *High, Default, Low* can map to:

- the ratio of sensor nodes that are awake: *80%, 20%, 5%*
- the delivered time-series data: *100%, 70%, 50%*
- the interval of data collection: *1 second, 10 seconds, 60 seconds*
- the frequency of data report: *Urgent, Periodical, Available*.

4.5 Event Detection

The current detection mechanism is based on an imperative algorithm, which is executed once every time instant (Fig.9). The main loop selects subexpressions dynamically and computes the current instance of a target composite event from the current primitive event and stored past information. For example, E denotes the event expression to be detected, and subexpressions of E are indexed 1 to k in bottom-up order. The operation result is $E^k (= E)$. Each operation in the expression needs its own indexed state variables (e.g., past events, time instant, and spatial information). For example, *Negation* $E = (A; B) - C_T$, an instance of $(A; B)$ is an instance of $(A; B) - C$ unless invalidated by some instance of C . Thus, if the current instance of $(A; B)$ is valid, the instance of C with maximum start time can be the only one to invalidate. *Conjunction* requires storage of the instance with maximum start time from each subexpression. If the event expressions are known before the execution, a canned detection component can be created for common use. The formal proof of the algorithm is out of scope of this paper. We implemented a prototype based on a simple automata with support of parameterized values and time constraints.

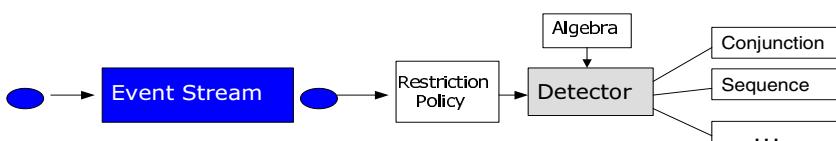


Fig. 9. Event Detection

5 Temporal Ordering

Sensor networks are used to monitor real world phenomena and for such monitoring applications, physical time is crucial. In global computing environments, such sensor data flow over heterogeneous networks and ordering events is difficult. We cannot assume a global clock, or globally synchronized physical clocks, to correlate events. Moreover when the store-and-forward paradigm is used for communication, message propagation delay is unavoidable. Traditional message ordering based on a transport layer protocol is not applicable.

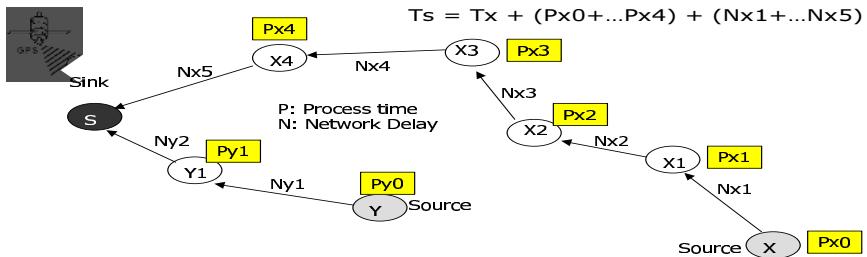


Fig. 10. Lightweight Local Clock Propagation

Thus, we use timestamps embedded in events for correlation, which provide a real-time mechanism. Temporal ordering of events is highly influenced by the event detection method, timestamping methods and the underlying time systems. In this section, we describe our time system for timestamping primitive events. Temporal ordering is based on interval semantics and is described precisely in the Appendix.

In many real world scenarios, wireless networks may be deployed with relay nodes to the Internet and it is possible that relay nodes can connect to GPS. This makes us consider the use of GPS in distributed systems, including wireless network environments. However, GPS is not suitable for use in a large class of smart devices due to its high power consumption and the required line of sight to satellites. But GPS may be the key for providing accurate time adjustment at certain nodes that are less resource constrained within wireless ad hoc networks. We define two categories of network environment; where NTP is deployed with GPS, such as the Internet domain, and where networks are isolated in ad hoc mode without GPS or any other deterministic time synchronization mechanism. For the first category, we use interval-based point timestamps for primitive events (see section 3), where the interval low and high end values are computed as described in [16] to allow for clock uncertainty and network delay. For timestamping composite events we use interval-based semantics, unlike [16] where a new timestamp is taken on the detection of a composite event. For the second category, several time synchronization mechanisms have been proposed (see section 6). Among those, we investigated the one described in [25]. The idea of the algorithm is not to synchronize the local computer clocks of the devices but instead to generate timestamps from a local clock. When such locally generated timestamps are passed between devices, they are transformed to the local time of the receiving device. As a result of the experiments in [29], we propose a simplified protocol *Lightweight Local Clock Propagation* explained below.

Thus our proposal is a coordinated approach with and without the use of GPS. We use interval-based timestamps throughout, with interval-based point timestamps for primitive events and interval-based semantics for composite events. Sensor events could be aggregated at gateway nodes with transformed timestamps and passed towards a subscriber node in the Internet environment, where GPS-based time synchronization is deployed.

Lightweight Local Clock Propagation: is on-demand based timestamp synchronization. Fig.1 shows the operation from source nodes X and Y to the sink node S. The basic idea is that each node calculates its processing time using a local clock, and at the sink node, the sum of the processing times is subtracted from the event arrival time to estimate the occurrence time. Comparable timestamps are therefore created at sink nodes instead of network-wide. This requires the following two assumptions:

- Network delay is negligible, where the node is close to the radio or network deployment is dense. Thus, $O(\text{nanoseconds})$ over a path to the sink of $O(100s)$ meters..
- Clock drift is negligible, where the node carries an oscilloscope that guarantees less than 10 ppm drift. One part per million (ppm) corresponds to 1 second in 11.5 days.

The detail of algorithms and experiments are out of the scope of this paper.

6 Related Work

Much composite event detection work has been done in active database research. SAMOS [7] uses Petri nets, in which event occurrences are associated with a number of parameter-value pairs. An early language for composite events follows the Event-Condition-Action (ECA) model and resembles database query algebras with an expressive syntax. Snoop [4] is an event specification language for active databases, which informally defines event contexts. The transition from centralized to distributed systems led to the need to deal with time. [4] presents an event-based model for specifying timing constraints to be monitored, and to process both asynchronous and synchronous monitoring of real-time constraints. [18] proposes an approach that uses the occurrence time of various event instances for time constraint specification. GEM [20] allows additional conditions, including timing constraints, to combine with event operators for composite event specification. In event-base middleware, publish/subscribe can provide subscription to composite events instead of leaving it to the client to subscribe to and correlate multiple primitive events. This reduces the communication within the system and potentially gives a higher overall efficiency, which is addressed in [23]. Hayton et al. [11], on composite events in the Cambridge Event Architecture [2], describe an object-oriented system with an event algebra that is implemented by nested push-down FSA to handle parameterized events. [12] provides time-restricted sequence and conjunction and a construct to detect the i^{th} occurrence of a given event.

Temporal message ordering has been an issue in traditional networks such as for system monitoring and in distributed event systems. In existing systems, the semantics of event order often depends on the application logic. Even the established Java Message Service (JMS) only guarantees the event order within a session where a session is a single-threaded context that handles message passing.

For real-time support, a common solution in wired networks provides a virtual global clock that bounds the value of the sum of precision and granularity within a few milliseconds. The following approaches aim to support a real-time mechanism. The 2g-Precedence model is enhanced for distributed event ordering and composite event detection using 2g-precedence-based sequence and concurrency operators [26]. Events from different sites can only be ordered if they are at least two clock ticks apart. The 2g-precedence model is applicable for closed networks with interconnected servers. Network Time Protocol (NTP) is an Internet standard that supports the assignment of real-time timestamps with given maximal errors. However, in open distributed environments, not all servers are interconnected and event ordering based on NTP may lead to false event detection. Interval-based time systems define event order based on intervals. In [16], timestamps of events can be related to UTC (Universal Coordinated Time) with bounded accuracy, and event timestamps are modeled using accuracy intervals. They use NTP that provides reference time injected by a GPS time server and, in addition, returns reliable error bounds. In [23], the interval timestamp model introduced in [16] is adopted.

For wireless network environments, [24] presents a GPS based virtual global clock, which is used for timestamping events, and deploys a similar concept to 2g-precedence. Without the existence of GPS, there is no means of synchronizing the clocks of all the nodes in a deterministic fashion with an upper bound independent of the message propagation delay and system size. Logical time cannot be used to determine temporal ordering, because causal ordering of events in the real world must be maintained. Thus, physical time has to be used, requiring clock synchronization. However, most of the synchronization algorithms rely on partitioned networks. Post-facto synchronization [6] is based on unsynchronized local clocks but limits synchronization to the transmit range of the mobile nodes. In [25], they take a similar approach of unsynchronized clocks.

7 Conclusions and Future Work

In this paper, we introduce a generic composite event semantics, which combines traditional event composition and the generic concept of data aggregation in wireless sensor networks. The main focus is on supporting time and space-related issues such as temporal ordering, duplicate handling, and interval-based semantics, especially for wireless network environments. Our approach includes definition of precise and complex temporal relationships among correlated events using interval-based semantics. Our event correlation semantics supports a new paradigm coming from the recent evolution of ubiquitous computing with a rapid increase of event monitoring capability by wireless devices, requiring more sophisticated event correlation over time and space. Work is ongoing on high level language definition and the transformation of event algebra, so that complex expressions can be more efficiently implemented in resource constrained devices over mobile ad hoc networks. We are working on a complete implementation, including various timestamping environments and parallel/hierarchical composition. Integration with WSN middleware such as TinyLIME is in progress.

Acknowledgment. This research is funded by EPSRC (Engineering and Physical Sciences Research Council) under grant GR/557303.

References

1. Allen, J. et al. Maintaining Knowledge about Temporal Intervals. *CACM*, 26(11), 1983.
2. Bacon, J. et al. Generic Support for Distributed Applications. *IEEE Computer*, 68-77, 2000.
3. Chaintresu, A. et al. Pocket Switched Networks: Real-world mobility and its consequences for opportunistic forwarding. *Technical Report, University of Cambridge*, 617, 2005.
4. Chakravarthy, S. et al. Snoop: An expressive event specification language for active databases. *Data Knowledge Engineering*, 14(1), 1996.
5. Curino, C. et al. TinyLIME: Bridging Mobile and Sensor Networks through Middleware. *Proc. PerCom*, 2005.
6. Elson, J. et al. Wireless Sensor Networks: A New Regime for Time Synchronization. *Workshop on Hot Topics in Networks (Hotnets-I)*, 2002.
7. Gatziu, S. et al. Detecting Composite Events in Active Database Systems Using Petri Nets. *Proc. RIDE-AIDS*, 1994.
8. Geppert, A. et al. Event-based distributed workflow execution with EVE. *Technical Report Univ. Zurich*, 1996.
9. Gomez, R. et al. Durative Events in Active Databases. *Proc. ICEIS*, 2004.
10. Gruber, B. et al. The architecture of the READY event notification service. *Proc. ICDCS*, 1999.
11. Hayton, R. et al. OASIS: An Open architecture for Secure Inter-working Services. *PhD thesis, Univ.of Cambridge*, 1996.
12. Hinze, A. et al. A flexible parameter-dependent algebra for event notification services. *Tech. Report, FU Berlin*, 2002.
13. Hoffmann-Wellenhof, B.H. et al. GPS: Theory and Practice. *Springer*, 1994.
14. Kopetz, H. et al. Real-time systems development: The programming model of MARS. *Proc. ISADS*, 1993.
15. Lazaridis, I. et al. Capturing Sensor-Generated Time Series with Quality Guarantees. *Proc. ICDE.*, 2003.
16. Liebig, C. et al. Event Composition in Time-dependent Distributed Systems. *Proc. 4th CoopIS*, 1999.
17. Liu, G. et al. Composite Events for Network Event Correlation. *Proc. IFIP/IEEE International Symposium on Integrated Network Management*, 1999.
18. Liu, G. et al. A Unified Approach for Specifying Timing Constraints and Composite Events in Active Real-Time Database Systems. *Proc. IReal-Time Technology and Applications Symposium*, 1998.
19. Madden, S. et al. TAG: A tiny aggregation service for ad-hoc sensor networks. *Proc. of Operating Systems Design and Implementation*, 2002.
20. Mansouri-Samani, M. et al. GEM: A Generalized Event Monitoring Language for Distributed systems. *IEE/IOP/BCS Distributed systems Engineering Journal*, 4(2), 1997.
21. Ma, C. et al. COBEA: A CORBA-based event architecture. *Proc. COOTS*, 1998.
22. Muehl, G. et al. Filter similarities in content-based publish/subscribe systems. *Proc. ARCD*, 2002.
23. Pietzuch, P. Shand, B. and Bacon, J. Composite Event Detection as a Generic Middleware Extension. *IEEE Network Magazine, Special Issue on Middleware Technologies for Future Communication Networks*, 2004.

24. Prakash, R. et al. Causality and the Spatial-Temporal Ordering in Mobile Systems. *Mobile Networks and Applications*, 9(5):507-516, 2004.
25. Roemer, K. Time Synchronization in Ad Hoc Networks. *ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc01)*, 2001.
26. Schwiderski, S. Monitoring the Behavior of Distributed Systems. *PhD thesis, University of Cambridge*, 1996.
27. Yoneki, E. and Bacon, J. Distributed Multicast Grouping for Publish/Subscribe over Mobile Ad Hoc Networks. *Proc. IEEE WCNC*, 2005.
28. Yoneki, E. and Bacon, J. Determination of Time and Order for Event-Based Middleware. *Proc. PerCom-MP2P*, 2005.
29. Yoneki, E. and Bacon, J. Event Order with Interval Timestamp in Event Correlation Service over Wireless Ad Hoc Networks. *Companion Proc. Middleware*, 2004.

Appendix: Interval Semantics - Timestamp for Composite Events

Table 3. Interval Semantics - Timestamp for Composite Events

Relation	Timestamps of Primitive Events	Point	Interval	Interval/Point	Point/Interval
1 A before B (A + B) (A B) (A ; B)	P-P: $t_p(A) < t_p(B)$ I-I: $t_i(A)^h < t_i(B)^l$ I-P: $t_i(A)^h < t_p(B)$ P-I: $t_p(A) < t_i(B)^l$	○ A ○ B ● ● ● ● ● ●	○—A—○ ○—B—○ ● —● —● ● —● —● ● —● —●	○—A—○ ○ B ● —● —● ● —● —● ● —● —●	○ A ○—B—○ ● —● —● ● —● —● ● —● —●
2 A meets B * (A + B) (A B) (A B) (A ; B) ₀	P-P: NA I-I: $t_i(A)^h = t_i(B)^l$ I-P: $t_i(A)^h = t_p(B)$ P-I: $t_p(A) = t_i(B)^l$		○—A—○ ○—B—○ ● —● —● ● —● —● ● —● —●	○—A—○ ○ B ● —● —● ● —● —● ● —● —●	○ A ○—B—○ ● —● —● ● —● —● ● —● —●
3 A overlaps B (A + B) (A B) (A B)	P-P: NA I-I: $(t_i(A)^l < t_i(B)^l) \wedge (t_i(A)^h > t_i(B)^l)$ I-P: NA P-I: NA		○—A—○ ○—B—○ ● —● —● ● —● —● ● —● —●		
4 A finishes B (A + B) (A B) (A B)	P-P: NA I-I: $(t_i(A)^l < t_i(B)^l) \wedge (t_i(A)^h = t_i(B)^h)$ I-P: $t_i(A)^h = t_p(B)$ P-I: $t_p(A) = t_i(B)^h$		○—A—○ ○—B—○ ● —● —● ● —● —● ● —● —●	○—A—○ ○ B ● —● —● ● —● —● ● —● —●	○ A ○—B—○ ● —● —● ● —● —● ● —● —●
5 A includes B (A + B) (A B) (A B)	P-P: NA I-I: $(t_i(A)^l < t_i(B)^l) \wedge (t_i(A)^h > t_i(B)^h)$ I-P: $(t_i(A)^l < t_p(B)) \wedge (t_i(A)^h > t_p(B))$ P-I: NA		○—A—○ ○—B—○ ● —● —● ● —● —● ● —● —●	○—A—○ ○ B ● —● —● ● —● —● ● —● —●	
6 A starts B (A + B) (A B) (A B)	P-P: NA I-I: $(t_i(A)^l = t_i(B)^l) \wedge (t_i(A)^h < t_i(B)^h)$ I-P: $t_i(A)^l = t_p(B)$ P-I: $t_p(A) = t_i(B)^l$		○—A—○ ○—B—○ ● —● —● ● —● —● ● —● —●	○—A—○ ○ B ● —● —● ● —● —● ● —● —●	○ A ○—B—○ ● —● —● ● —● —● ● —● —●
7 A equals B (A + B) (A B) (A B)	P-P: $t_p(A) = t_p(B)$ I-I: $(t_i(A)^l = t_i(B)^l) \wedge (t_i(A)^h = t_i(B)^h)$ I-P: NA P-I: NA	○ A ○ B ● ● ● ● ● ●	○—A—○ ○—B—○ ● —● —● ● —● —● ● —● —●		

- depicts the timestamp for the composite events
 - * A meets B where $t(A)^h$ and $t(B)^l$ share the same time unit
- $t(A)$: timestamp of an event instance A
- $t_p(A)$: Point-based timestamp
- $t_i(A)_l^h$: Interval-based timestamp from event composition
- $t_{pi}(A)_l^h$: Point-interval-based timestamp
(compared same as point-based timestamp but using interval comparison)

- P-P: Between Point-based and Point-based timestamps
 I-I: Between Interval-based and Interval-based timestamps
 I-P: Between Interval-based and Point-based timestamps
 P-I: Between Point-based and Interval-based timestamps

Real-time Period T :

$$[t(A)^l, t(A)^h] - [t(B)^l, t(B)^h] < T = \begin{cases} YES : max(t(B)^h, t(A)^h) - min(t(B)^l, t(A)^l) \leq T(1 - \rho) \\ NO : max(t(B)^l, t(A)^l) - min(t(B)^h, t(A)^h) < T(1 + \rho) \\ MAYBE : otherwise \end{cases}$$

where ρ is maximum clock skew.

Semantic-Based Matching and Personalization in FWEB, a Publish/Subscribe-Based Web Infrastructure

Simon Courtenage and Steven Williams

University of Westminster,
115 New Cavendish Street, London, United Kingdom
`{courtes, williast}@wmin.ac.uk`

Abstract. The web is a vast graph built of hundreds of millions of web pages and over a billion links. Directly or indirectly, each of these links has been written by hand, and, despite the amount of duplication among links, is the result of an enormous effort by web authors.

One has to ask if it is possible that some of this labour can be automated. That is, can we automate some of the effort required to create and maintain links between pages? In recent work, we described FWEB, a system capable of automating link creation using publish/subscribe communication among a peer-to-peer network of web servers. This allowed web servers to match information about link requirements and page content in circumstances where we specify an anchor in terms of what content we want to link to, rather than a specific URL. When such a match is successful, a link between the pages is automatically created.

However, this system relied on simple keyword-based descriptions, and has several drawbacks, verified by experiment. In this paper, we show how the use of shared ontologies can improve the process of matching the content requirements for links and the descriptions of web pages. We report on our experience of using FWEB and, in addition, show how the capabilities of the FWEB architecture can be extended to include link personalization and explicit backlinks.

1 Introduction

The web is a vast repository of hypertext documents and other resources, connected by hyperlinks. Pages on the web number in hundreds of millions, while the number of hyperlinks is estimated at more than a billion [11]. While the content of some web pages may be generated automatically, each hyperlink is essentially hand-written (either directly into the page or stored in a database from which a page is generated). The structure of the web, therefore, is testament to a truly enormous effort by web page authors.

Creating and maintaining relevant and appropriate hyperlinks requires some effort on the part of the page author, particularly if the hyperlink is to add some value to the page being written. When web page authors are writing or

editing web documents, they insert links to other web pages that are known to them and which appear to offer content that is relevant to part of the page they are writing/editing. The part of this process that is problematic is how these other pages become known to the web page author. It may be that the author is also the author of these other pages, or that they are part of the same web site (for example, as in intra-website navigation bars). There is also the case where the author wishes to clarify some point in their web document and wishes to locate information elsewhere on the web that will provide that clarification. How, in this case, does the author locate a relevant page? There seem to be two possibilities: (i) they already know of its existence from their own web surfing experience, or (ii) they find it through a search engine. What is common to both of these possibilities is that

- the page to link to must already exist, in order for the URL to be written into the referring page,
- if there is a choice of pages to link to, only one can be chosen (although there is the possibility of repeating the anchor with a different URL, this effectively replicates the problem)

In [3], we examined whether the process of building the web can be automated under certain conditions. In other words, can the creation of links between pages be performed automatically? One of the benefits of being able to automate this process is that in fast-moving and incomplete spheres of knowledge, manually maintaining an up-to-date set of hyperlinks to new and constantly changing information is time-consuming and costly. We showed that this can be done provided that, in place of actual URLs, we specify a hyperlink in terms of what kind of content we want to link to and then use a publish/subscribe form of communication to match up what we are interested in with what kind of content a page offers. To demonstrate this concept, we described an experimental web architecture, called FWEB, in which web servers were linked in a peer-to-peer network with publish/subscribe communication to exchange and match subscriptions for links and publications of page summaries.

However, the FWEB architecture in [3] has been found to have certain limitations. While the basic infrastructure allowed hyperlinks to be established automatically using the publish/subscribe mode of communication, it relied on simple free-form keywords, which, we have found in practice, produce few meaningful matches between pages.

In this paper, we report on how this problem has been addressed in the latest version of FWEB. We show how the use of the taxonomic elements of user-selected ontologies can improve the matching process. This allows the page author to describe the content of a link or the summary of a page using terms from a relevant ontology, and includes simple matching against the hierarchy of terms in the ontology to resolve undefined references. The benefit of this work is to resolve many of the problems in the matching of subscriptions for links and publications of page data, whilst introducing a more powerful and expressive means of doing so. In addition, we describe extensions to the basic publish/subscribe protocol used to

automatically find hyperlinks, to allow users to personalize their view of the web and to create explicit bi-directional links between pages that allow users to navigate forwards and backwards across hyperlinks.

The structure of the rest of this paper is as follows: in Sections 2 and 3, we review publish/subscribe communication and the FWEB infrastructure presented in [3]; in Section 4, we explain the problem of matching subscriptions and publications in FWEB and present a solution based on ontologies; in Section 5, we describe how FWEB has been extended to allow users to personalize their view of the web graph and to allow bi-directional navigation of hyperlinks. Finally, Section 6 presents related work and Section 7 presents conclusions and areas of further work.

2 Publish/Subscribe Communication

Publish/Subscribe systems [7] form an important communications paradigm in distributed systems, one in which servers (or producers of messages) are decoupled from clients (or consumers) by the network. Instead of clients contacting servers directly to request services or information, clients register a subscription with the network to receive messages satisfying certain criteria. Servers publish information onto the network, without knowing who will receive it, and the network undertakes to route messages to the appropriate clients based on the set of subscriptions currently in effect.

Within the publish/subscribe paradigm, there are many different types of systems based on how a subscriber makes a subscription. Traditional publish/subscribe systems create channels, groups or topics, sometimes hierarchical, under which messages may be classified. In this case, a subscription is simply the identity of the channel, group, or topic that a user wants to receive messages from. Once subscribed, the user or subscribe receives all messages that are published under that channel, group or topic. Recently, another approach to publish/subscribe has been developed that allows subscriber to specify their interests in terms of the kind of *message content* they want to receive: this is known as content-based routing.

The advantage of combining content-based routing and publish/subscribe, to create *content-based publish/subscribe* [2] [1] [13], over more conventional systems is the far greater flexibility that is permitted in creating subscriptions. Subscribers are in effect allowed to create their own message groupings rather than simply sign up to predefined ones. A subscriber can, for example, request to receive all messages of a certain type, such as StockQuote messages (perhaps from a particular publisher), where the company name in the StockQuote message is "IBM". When the subscription has been registered with the network (typically a network of servers overlaying a TCP/IP-based network), the network undertakes to route to the subscriber all messages of that type whose content satisfies the subscriber's criteria, typically using an overlay network of *brokers*, servers whose role is to match up subscriptions with publications.

3 FWEB Infrastructure

In [3], we proposed a new web infrastructure, called FWEB, that allows page authors to specify hyperlinks, not in terms of specific URLs, but in terms of the content of the pages to be linked to. The aim of FWEB is to allow automatic creation of hyperlinks as and when matching content becomes available. In areas where knowledge may be incomplete and expanding, the process of manually locating new or updated pages that are relevant to be the target of hyperlinks is an expensive and time-consuming process. The benefit of FWEB is to reduce the cost of this process by automating it.

In FWEB, therefore, we indicate what content we want to link to, rather than what URL. Through the use of new tags for FWEB, a web document author can indicate where new links should be inserted into a document as and when new and relevant content is found by using a `<LINKTO KEY="..."> ... </LINKTO>` tag, rather than an HTML anchor tag. The `<LINKTO>` tag is used around text which the author would like to act as a hyperlink to other documents (that may or may not exist in the web graph when the document is created). The `KEY` attribute of the tag contains those keywords to be used in finding matching content on other pages. For example

```

<HTML> <HEAD>
<TITLE>Web Servers</TITLE>
<SUMMARY> Web servers </SUMMARY>
</HEAD>
<BODY>
<H1> The role of web servers </H1>
The role of an
<LINKTO KEY="web servers"> HTTP-enabled web server </LINKTO>
is to respond to
<LINKTO KEY="HTTP">HTTP (Hypertext Transfer Protocol) </LINKTO>
requests.
<!-- rest of page -->
</BODY> </HTML>

```

We also include a new SUMMARY tag, to be included in the header of a page and which acts as an aid to matching page content to link requirements. The SUMMARY tag is similar to the use of the existing HTML `<META>` tag, which has been used in the past to advertise a page's content to search engines. We have defined a separate tag, rather than reuse `<META>`, in order to avoid confusion and also to allow future extension of use.

The syntax of the SUMMARY tag is:

```
<SUMMARY> keyword-list </SUMMARY>
```

and is placed between the `<HEAD>` and `</HEAD>` tags in an HTML document.

For example,

```
<HTML> <HEAD>
<TITLE>CGI Programming Notes</TITLE>
<SUMMARY> CGI, Web Server, Protocol, HTTP
</SUMMARY>
</HEAD>
<BODY>
<!-- body of web page -->
</BODY> </HTML>
```

The problem faced by FWEB is how to connect a web page with a `<LINKTO>` tag with a page with a `<SUMMARY>` tag when the keywords in the `<LINKTO>` tag and the content of the `<SUMMARY>` tag agree. We solve this problem by arranging FWEB servers as a broker network and employing the content-based publish/subscribe paradigm.

The infrastructure for FWEB is a P2P network of *brokers*, for which is currently based on CHORD [5] [16]. CHORD is a popular distributed P2P service with a single operation: node lookup based on Distributed Hash Tables (DHTs). Given a particular hash key, the CHORD architecture allows fast and efficient lookup of the P2P node associated with that particular key. Each FWEB server participates in the P2P system, acting not only as a web server, therefore, but also as a broker node in a content-based publish/subscribe system in order to link pages together.

Placing a new web page in the document root of an FWEB server triggers the server to parse the document and extract the content of any `<SUMMARY>` or `<LINKTO>` tags (currently, the parsing of new or updated pages is implemented in our prototype FWEB server using directory polling). If a `<LINKTO>` tag is found, then its keywords are used to create subscriptions. Each keyword is hashed using a hash function to locate the FWEB server in the CHORD-based P2P network which will act as broker for the subscription. The located server is then contacted with the details of the subscription (unhashed keyword and URL of the document

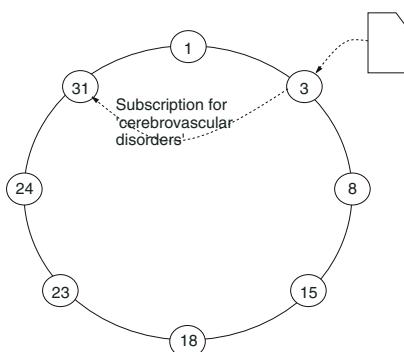
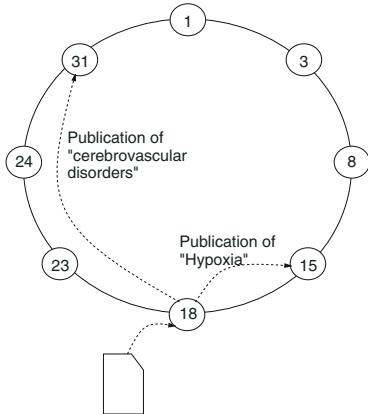
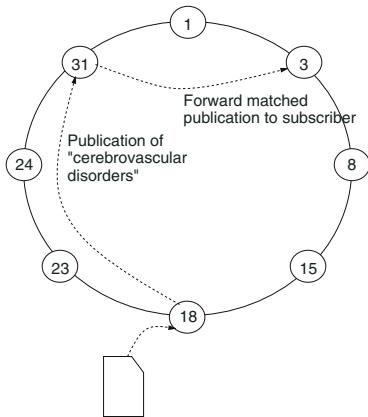


Fig. 1. Distributing subscriptions

**Fig. 2.** Distributing subscriptions**Fig. 3.** Forwarding matched publication

containing the <LINKTO> tag). Similarly, the keywords in the page summary are used to create publications, by hashing the keywords to locate the nodes in the FWEB P2P network that the publications should be sent to.

For example, given a new web page containing the following <LINKTO> tag

```
<LINKTO KEY="cerebrovascular disorders"></LINKTO>
```

the keyword phrase is hashed to locate the node that should receive the subscription for content that match this description. If a page is published with the <SUMMARY> tag

```
<SUMMARY>cerebrovascular disorders, Hypoxia </SUMMARY>
```

then publications are created for the keyword content.

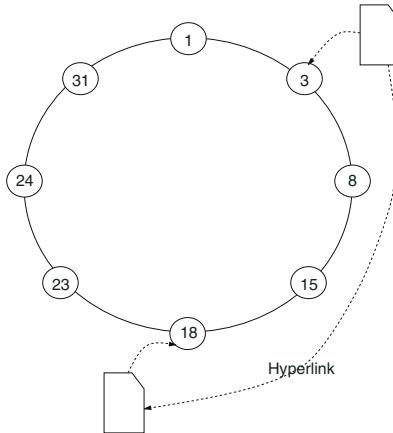


Fig. 4. Establishing hyperlink

When an FWEB server receives a subscription, it attempts to match the subscription against the web pages that it knows have already published summaries¹.

Since both the page and the subscription are sent to a node on the basis of hashing a keyword, then if they contain the same keyword, they will be sent to the same node. If the match is successful, then details of the publication are sent back to the FWEB server that made the subscription.

Using the example <LINKTO> and <SUMMARY> tags above, we show how subscriptions and publications are matched to create hyperlinks. In Figure 1, the placement of a new page under the FWEB server that is node 3 in the P2P network creates a subscription for "cerebrovascular disorders". Hashing the keyword phrase produces 31 as the id of the node acting as broker for the subscription. Figure 2 shows a similar process on the FWEB server at node 18, where a new page creates a publication based on the content of the <SUMMARY> tag in the page. The FWEB server at node 31, when it receives the publication, matches against the subscription received earlier. Hence the publication is forwarded to node 8, as in Figure 3. Finally, Figure 4 shows the establishment of a hyperlink between the page with the <LINKTO> tag and the page with the SUMMARY tag, once the publication is received by the node that originated the subscription.

In the current FWEB, a hyperlink is only accepted if its summary contains all the keyword phrases in a <LINKTO> tag (by collecting and collating the publications received). Effectively, therefore, the keywords in a <LINKTO> tag form a conjunction which must be satisfied for a link to be created. This mimics the basic structure of a query in a conventional search engine, and is intended to help restrict the number of returned search results.

¹ Because of this, FWEB is related to work on continuous querying in P2P systems, for example, [10].

In the current development of FWEB, the hyperlink is established by creating a file (with a .url file extension) associated with the original HTML file that stores the hyperlinks received for <LINKTO> tags. When the file is requested by a browser, the web server parses the HTML document and the file of received hyperlinks, replacing the <LINKTO> tags with hyperlink anchors. To do this, the current web server implementation redirects on every request to a Java Servlet that performs the parsing and replacement process, using a DHTML drop-down menu to display the multi-valued hyperlinks. This ensures compatibility with current web browsers. Figure 5 show a screenshot of a web page sent by an FWEB server with multi-valued hyperlinks displayed using a drop-down menu.

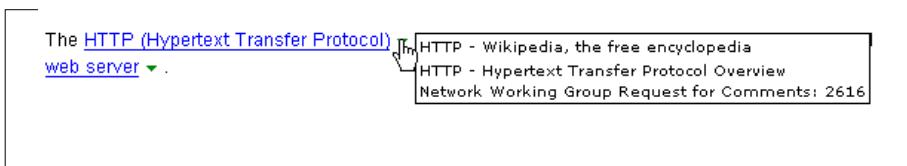


Fig. 5. Displaying a page with multi-valued hyperlinks

4 Semantic-Based Matching of Subscriptions and Publications

In this section, we describe the use of ontologies in the descriptions of anchors and page content and the implementation of semantic-based matching of subscriptions and publications for related terms in an ontology.

4.1 Problems of Keyword-Based Matching

In the version of FWEB described in [3], subscriptions for, and publications of, page content are made using free-form keywords, with no restriction on the keywords that can be used. This is similar to the way in which queries are posed to search engines, and suffers from many of the same problems.

For example, synonyms mean that the same content can be described using different keywords. FWEB, however, uses syntactic equivalence (based on the DHT-based implementation of the underlying P2P network), and coping with synonyms is therefore difficult. One possible solution would be a dictionary of available synonyms to be distributed among peers in the network to expand subscriptions to include all possible synonyms of a keyword in the subscription. However, maintaining the dictionary across a large number of peers would be complex, requiring a central source for the dictionary updates. In addition, different knowledge domains may impose different meanings on the same term, resulting in ambiguity if the knowledge domain is not specified.

Another problem is the semantics of the keywords used. Consider, for example, the term "Heart Disease". In the Unified Medical Language System (UMLS) [9], for example, the UMLS considers that "Myocardial Infarction" 'is-a' "Heart Disease" (where "is-a" represents the parent-child containment relationship in the UMLS ontology). If a page author wants to link from an anchor on "Heart Disease", then a page with content on myocardial infarction is a suitable candidate according to the UMLS. In the current FWEB prototype, this would require that the author of the page on myocardial infarction describe the page using the phrase "Heart Disease", as well as "Myocardial Infarction", so that FWEB can match the subscription for "Heart Disease" against the same publication. This is quite possible, but has its limitations, since it relies on the author of the page on myocardial infarction describing the page in a way that allows the match to succeed.

The free-form keyword approach works when it is clear which specific keywords should be used, so that exact matching can be performed using the hashing functions in the underlying P2P network. Unfortunately, this is unlikely to be often the case, especially in such a distributed environment, where page authors are essentially unknown to each other.

Relying on the keyword-only approach also does not allow the full expressiveness power of an ontology to be used. For example, "coronary artery disease" and "heart value disease" are also forms of heart disease, related via the is-a relationship, according to the UMLS. We might want to create a link from an anchor on *forms* of heart disease, rather than heart disease in general. Using the keyword approach, we would have to create subscriptions for each of the terms that are related via the "is-a" relationship to heart disease, in order to represent the relationship.

4.2 Specifying Ontology-Based Subscriptions

To solve these problems with keyword-based matching, therefore, we introduce the use of ontologies into the creation and matching of subscriptions and publications in FWEB. Not only will this remove any semantic ambiguities or confusion about which term to use to describe a link or page, but will also allow us to represent relations between terms, based on the hierarchical taxonomy between terms in an ontology.

Consequently, we have extended the syntax of the <LINKTO> tag to include ontological features. This allows the page author to describe the required content of pages to be linked using terms from an ontology selected for that purpose by the page author. The <LINKTO> tag also allows the author to identify the ontology being used. This, in turn, has meant extending the communications infrastructure of FWEB to take advantage of these features in matching subscriptions with publications. The new <LINKTO> syntax is

```
<LINKTO NS=". ." KEY=". ."> anchor text </LINKTO>
```

where the KEY attribute is a comma-separated list of *subscription values* in the following format (where { ... } indicates an optional element):

{ {*CHILD-TERM*} :*RELATIONSHIP* : } *PARENT-TERM* }

where *CHILD-TERM*, *PARENT-TERM* and *RELATIONSHIP* are defined specific to an ontology. The only mandatory component of a subscription value is *PARENT-TERM*; the specification of *RELATIONSHIP* and *CHILD-TERM* are optional. An example of a subscription value is

Myocardial Infarction :is-a :Heart Disease

This example subscription is effectively the same as a subscription value for "Myocardial Infarction" alone. However, we also allow the use of the special placeholder term '_' (underscore), which can match any term or relationship. For example

_:is-a :Heart Disease

will match against any term which is in an "is-a" relationship with "Heart Disease", such that the term matching '_' is the child term of the "is-a" relationship. This example subscription value will match all of "coronary heart disease", "myocardial infarction" and "heart value diseases". If we generalize the subscription further to

::Heart Disease

so that the relationship between the child and parent terms is also unspecified, then we add the term "Heart" to the list of child terms, since "Heart" is the child term in a "location-of" relationship with "Heart Diseases".

The remaining attribute of <LINKTO> is the NS, or namespace, attribute, which identifies the ontology to which the terms used in the KEY attribute belong. At present, we assign unique identifiers to distinct ontologies and use these as the values of the namespace attribute to allow us to specify which ontology is being used in the definition of the subscription values.

The syntax of the <SUMMARY> tag is also amended, albeit only to include the namespace attribute to identify the ontology to which the terms in the tag belong. The new syntax is:

<SUMMARY NS=".."> keyword-list </SUMMARY>

Keywords included between the opening and closing tags should be terms in the identified ontology.

One of the consequences of this approach is that subscriptions and publications will only match if both contain matching terms from the same ontology. Use of the same ontology by a particular community should lead, therefore, to the creation of a particular web subgraph for that community based on links created through FWEB.

4.3 Implementing Semantic-Based Matching

Implementing semantic-based matching of subscriptions and publications relies on the FWEB servers having access to the ontology in order to resolve the

subscriptions. Given the example subscription `_: is-a : Heart Disease`, an FWEB server must expand the `_` operator to all matching terms.

To implement semantic-based matching of subscriptions and publications, little change needs to be made to the current FWEB architecture. When a page containing a `<LINKTO>` tag is placed in the document repository of an FWEB server, the server extracts the values of the `NS` (namespace) and `KEY` attributes. It identifies the ontology, using the value of the namespace attribute, and matches the subscription values from the `KEY` attribute against the ontology to resolve instances of the `_` operator. Each child term found by matching against the ontology is then made a separate subscription request.

In our current development, we have used a set of web services to provide ontology support, rather than build the ontology and ontological support into each FWEB server. As a general infrastructure, FWEB is meant to be ontology-independent and to allow `<LINKTO>` tags to be defined using any available ontology. Hence the use of web services to separate out and provide ontological support. We do require, though, that FWEB servers are able to identify which web service to contact using the value of namespace identifiers. In our current design, this takes the form of a central registry web service. Although this does introduce a degree of centralization into FWEB, it should be pointed out that the use of such centralized web services only occurs at the point when new pages are added or existing pages are updated under an FWEB server. Also, since the ontological support provided by web services is not stateful, these services can be replicated in much the same way that DNS services are replicated, thus relieving some of the load.

Figure 6 illustrates the process involved in creating subscriptions when a new page is added to an FWEB server. This shows that adding semantic-based matching inevitably increases network traffic overhead. Since there is no real-time requirement in FWEB, we do not envisage that the extra network traffic will affect the overall performance of an FWEB server, particularly if the rate of addition of new pages and updates to existing pages is low. On the other hand, we do not know how the accumulative overhead of network traffic will affect the network as a whole. (This would require an extensive simulation study.)

In the current version of FWEB, we have used conjunction to match the free-form keyword-based subscription values of a `<LINKTO>` tag with received publications. In other words, a page only becomes a link for a `<LINKTO>` tag, if the keywords in its summary include *all* the keywords in the `<LINKTO>` tag. This is necessary because of the multiple ways in which a particular topic of information may be described. Using ontologies as a source for subscription values, however, removes this source of ambiguity. Hence, for the semantic-based FWEB, we currently use disjunction when creating links. The content of a page's summary tag needs to contain only one of the subscription values of a `<LINKTO>` tag in order for the link to be created. Hence if the `<LINKTO>` tag is

```
<LINKTO NS=".." KEY="_:is-a :Heart Disease"> Heart Diseases
</LINKTO>
```

and the summary tag of another web page is

```
<SUMMARY NS="..">Myocardial Infarction </SUMMARY>
```

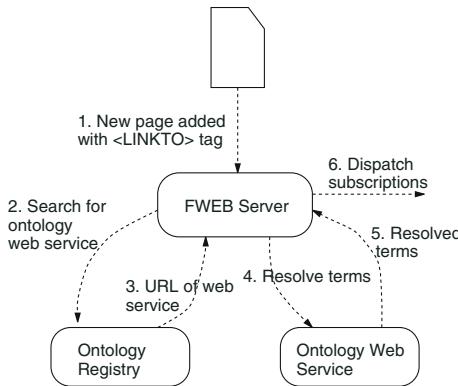


Fig. 6. Creating semantic-based subscriptions

a link will be created from the `<LINKTO>` tag to the page on myocardial infarction. This is, however, based on the rather restrictive assumption that each page can be described by just one ontological term.

5 Personalization and Backtracking in FWEB

In the current web, and in the current implementation of FWEB, there is only one globally-shared view of the web graph. A hyperlink from one page to another can be created without the permission of the author of the referred-to page, and is visible to all. Any visitor to a page can see and follow the links to other pages put in place by the page author.

FWEB, however, offers possibilities for personalization at group and individual level, as well as possibilities for permission-based link creation. These possibilities arise from the physical separation of the link values from the page they are used in and from the subscription/publication matching process necessary to create links.

5.1 Browser-Based Personalization

FWEB currently uses the HTTP protocol to send requested documents, including documents containing `<LINKTO>` tags. As mentioned earlier, before these documents are sent, the FWEB server replaces occurrences of `<LINKTO>` tags with matched URLs from a publications file associated with the document. When a web browser receives the document, therefore, the URLs matching the content of a `<LINKTO>` tag have been embedded in the HTML and the `<LINKTO>` tag is removed.

In the latest design for FWEB, however, we are experimenting with browser extensions that allow FWEB servers to send matched URLs separately from the documents whose `<LINKTO>` tags they have matched. The browser extensions

then perform the task of replacing the <LINKTO> tags with the URLs. The publications file of URLs is associated with the HTML document in the same way that external Cascading Stylesheet files are associated with an HTML document, using a header-located <LINK> tag referring to a URL for the relevant file. We use a custom-defined value for the REL attribute of the <LINK> tag to designate the file referred to as being a file of matched URLs. For example, the following link tag

```
<LINK REL="URL-Publication" HREF="index.url">
```

is used to load URLs to replace <LINKTO>tags in the HTML document.

When the page is loaded and displayed, along with the data from the associated publications file, the browser allows the user to right-click and mark a link in an FWEB-produced multi-valued hyperlink as not needed. This results in the addition of the URL to a list of 'excluded' URLs (which are stored in a local file). When the page is requested again, at some later time, the browser remembers that the link has been marked not for redisplay and consequently it is omitted from the rendered page.

5.2 Setting and Following Explicit Backlinks

A hyperlink is effectively created when a subscription successfully matches a publication, particularly since, in the redesigned FWEB, we consider the disjunction of subscriptions. Therefore, as well as notifying the subscribing FWEB server of the match by forwarding the publication data, we can also notify the publishing FWEB server. This allows a document to keep track of pages that link to it. We have designed a scheme that allows web pages under FWEB servers to maintain a list of URLs of pages with <LINKTO> tags that include its own URL in their associated URL files, and then to display the backlinks when loaded into a standard browser.

6 Related Work

There has been a great deal of work in designing and developing content-based publish/subscribe systems, for example, [2] [4] [8] [13] [15] [17], which have successfully tackled many of the problems involved in routing a message from its source to a client based solely on its content. Our work makes use of this research in its use of a broker-style network to mediate between subscriptions and publications, in a manner similar to Hermes [13]. However, content-based publish/subscribe systems, including Hermes, have mostly been used to implement distributed event notification systems, which aim to detect events and send notifications concerning events to interested parties as and when they arise. Event notification systems do not typically maintain event histories or state. However, FWEB requires that broker nodes keep track of past publications in order to create hyperlinks to documents that already exist. Moreover, we do not currently require filters in subscriptions more complex than simple equality.

Lewis *et al* [12] describe the use of a content-based publish/subscribe system as part of a complete semantic-based information dissemination system, allowing client browsers to make subscriptions to create information spaces, and then receive notifications when new information becomes available. Our system differs from this work in that the role of publish/subscribe in our case is simply to create ordinary hyperlinks between HTML-compliant web pages based on matching content on the basis of communication between web servers, which is a much simpler proposition. Moreover, our system is currently fully compatible with the current web. FWEB servers are ordinary web servers with additional functionality, and ordinary web browsers can (with a little added functionality) view web pages from FWEB servers. The system described in [12], however, depends heavily on Semantic Web markup and associated technologies, and would require custom-written browser support.

In terms of P2P systems, Chord has been used before to implement publish/subscribe systems: for example, content-based publish/subscribe systems using Chord is described in Triantafillou *et al* [19] and Terpstra *et al* [18]. The primary goals of [18] are the robustness of the routing strategy for content-based publish/subscribe and the implementation of a filtering strategy on top of a DHT-based P2P system, neither of which are currently handled by FWEB (FWEB's current concept of filters is limited to simple keyword equality which does not pose a problem in DHT-based P2P systems). In [19], the concern is with how range predicates (more complex filters on content, such as *less-than* or *greater-than*) can be implemented in DHT-based P2P systems such as CHORD where the use of hashing to locate nodes makes support of filters other than equality difficult.

P2P systems have also been used for semantic-based knowledge management. For example, Ehrlig *et al* [6] describe SWAP, a project to query distributed RDF collections in a P2P network. In Schlosser *et al* [14], a CHORD-like P2P topology is used to deploy a network of Semantic Web service providers and implement an efficient search mechanism without the need for centralized registries such as UDDI. The search mechanism in these works is a conventional P2P-based message broadcast mechanism: a peer wishing to search for a resource broadcasts a message to other peers and receives back responses from peers with matching resources. FWEB has a different concept of search. The use of a content-based publish/susbcrite mechanism means that there is no need to broadcast queries among peers. Also, the results of a search are inherently shareable in FWEB, since they lead to the creation of hyperlinks in a global hyperlinked document repository that is separate from the P2P network. In [6] and [14], the search results are for the client only.

7 Conclusion and Further Work

This paper has described the use of a P2P-based content-based publish/subscribe system to augment the current web, automating the process of hyperlink creation. It allows web authors to specify what kind of pages they want to link

to, rather than explicit URLs. The advantage of this approach is that links to matching pages are automatically added, maintained and updated without intervention from the web page author. This may suit knowledge domains where information is incomplete and expanding.

We have experimented with a prototype FWEB system, using 4-5 FWEB servers. The FWEB servers are written in Java and use JXTA to create the CHORD P2P ring, while documents in the system use free-form keywords to create subscriptions and publications. Standard browsers, such as IE, are used to request documents and display the results. We have found, in this small setting, that FWEB performs as we expected, and that hyperlinks are created for <LINKTO> tags as and when documents are added to FWEB servers whose publications of summary content matches their subscriptions. However, we have not as yet been able to test or simulate the behaviour of FWEB in a large-scale setting. The impact of the role of FWEB servers as brokers in a P2P publish/subscribe system on their performance as web servers needs to be investigated. If there is any detrimental affect, then the two roles may need to be separated, as in, for example, web servers and servlet engines.

There are a number of other areas of further work, which we describe briefly below.

Subscription languages. The current subscription language used to create subscription values for the KEY attribute of <LINKTO>tags is very basic. It doesn't, for example, allow a mix of complex conjunctive and disjunctive terms. As such, it is less expressive than the conventional boolean terms used by standard search engines. We plan to revisit this area of work to investigate how the expressiveness of subscription values can be enhanced.

Page Authoring. The design of FWEB, as presented in this paper, means that page authoring is more demanding. Adding <LINKTO>tags with correct and correctly-formatted KEY attributes is more time-consuming, and requires web page editors with built-in ontological support. Authors will need some guidance concerning the meaning of subscription values, which should be provided by the editing package.

Navigation. is more powerful than the current web, with the result, perhaps, that it is easier to get lost. Consequently, more support is required to allow users to make use of this expressiveness. For example, browser extensions to correctly display multi-valued hyperlinks, rather than the current Java servlet/DHTML-based ad-hoc solution currently employed. This would also make the display of explicit backlinks possible.

Access-based URL permissioning. We plan to experiment with adding access-based permissioning to the matching of subscriptions and publications in FWEB. At present, a hyperlink is effectively created if a publication matches a subscription, but it would be possible to make the notification of the successful match contingent on the permission of the author of the page that made the

publication. We plan to experiment with different permissioning frameworks, such as tokens, to see if this is feasible.

Link verification FWEB provides a means to deal with dead links (links to pages that no longer exist), just as it provides a means of implementing hyperlink permissioning. This would depend on an FWEB server being notified when a page in its collection is deleted and publishing a delete notification onto the P2P network. In general, there is a requirement for automatic link maintenance in FWEB that is derived from the implementation of automatic link creation.

Acknowledgements

We gratefully acknowledge the support of the UK Engineering and Physical Sciences Research Council (under EPSRC Grant GR/S01573/01) for this work.

References

1. A. Carzaniga, D. Rosenblum, and A. Wolf. Content-based addressing and routing: A general model and its application, 2000.
2. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, Aug. 2001.
3. S. Courtenage and S. Williams. Automatic hyperlink creation using p2p and publish/subscribe. In *Workshop on Peer-to-Peer and Agent Infrastructures for Knowledge Management (PAIKM)*, Kaiserlautern, Germany, April 2005.
4. S. A. Courtenage. Specifying and detecting composite events in content-based publish/subscribe systems. In *1st International Workshop on Discrete Event-Based Systems*, jun 2002.
5. F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan. Building peer-to-peer systems with Chord, a distributed lookup service. In IEEE, editor, *Eighth IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII). May 20–23, 2001, Schloss Elmau, Germany*, pages 81–86, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 2001. IEEE Computer Society Press.
6. M. Ehrig, P. Haase, F. van Harmelen, R. Siebes, S. Staab, H. Stuckenschmidt, R. Studer, and C. Tempich. The swap data and metadata model for semantics-based peer-to-peer systems. In *First German Conference on Multiagent Technologies (MATES-2003)*, Sept. 2003.
7. P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe, 2001.
8. F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Efficient matching for content-based publish/subscribe systems. Technical report, INRIA, 2000. <http://wwwcaravel.inria.fr/pereira/matching.ps>.
9. B. Humphreys and D. Lindberg. The umls project: making the conceptual connection between users and the information they need. *Bulletin of the Medical Library Association*, 81(2):170–177, 1993.

10. S. Idreos, M. Koubarakis, and T. Tryfonopoulos. P2P-DIET: an extensible P2P service that unifies ad-hoc and continuous querying in super-peer networks. In G. Weikum, A. C. König, and S. Dessloch, editors, *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD-04)*, pages 933–934, New York, June 13–18 2004. ACM Press.
11. J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins. The Web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science*, 1627:1–??, 1999.
12. D. Lewis, K. Feeney, K., T. Tiropanis, and S. Courtenage. An active, ontology-driven network service for internet collaboration. In *Workshop on Application of Semantic Web Technologies to Web Communities (SWWC) at ECAI'04*, Aug. 2004.
13. P. R. Pietzuch and J. M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *Proc. of the 1st Int. Workshop on Distributed Event-Based Systems (DEBS'02)*, pages 611–618, Vienna, Austria, July 2002.
14. M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. A scalable and ontology-based p2p infrastructure for semantic web services. In *Second IEEE International Conference on Peer-to-Peer Computing (P2P2002)*, 2002.
15. B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUUG'97*, 1997.
16. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *Proceedings of the ACM SIGCOMM*, pages 149–160, Aug. 2001.
17. R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward. Gryphon: An information flow based approach to message brokering, 1998.
18. W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–8. ACM Press, 2003.
19. P. Triantafillou and I. Aekaterinidis. Content-based publish/subscribe over structured p2p networks. In *1st International Workshop on Discrete Event-Based Systems*, May 2004.

A Cooperative Model for Wide Area Content Delivery Applications

Rami Rashkovits and Avigdor Gal

Technion Israel Institute of Technology,

Technion City, Haifa 32000, Israel

`ierami@tx.technion.ac.il, avigal@ie.technion.ac.il`

Abstract. Content delivery is a major task in wide area environments, such as the Web. Latency, the time elapses since the user sends the request until the server’s response is accepted is a major concern in many applications. Therefore, minimizing latency is an obvious target of wide area environments and one of the more common solutions in practice is the use of client-side caching. Collaborative caching is used to further enhance content delivery, but unfortunately, it often fails to provide significant improvements. In this work, we explore the limitations of collaborative caching, analyze the existing literature and suggest a cooperative model for which cache content sharing show more promise. We propose a novel approach, based on the observation that clients can specify their tolerance towards content obsolescence using a simple-to-use method, and servers can supply content update patterns. The cache use a cost model to determine which of the following three alternatives is most promising: delivery of a local copy, delivery of a copy from a cooperating cache, or delivery of a fresh copy from the origin server. Our experiments reveal that using the proposed model, it becomes possible to meet client needs with reduced latency. We also show the benefit of cache cooperation in increasing hit ratios and thus reducing latency further. Specifically, we show that cache collaboration is in particular useful to users with high demands regarding both latency and consistency.

1 Introduction

Content delivery is one of the major tasks in wide area environments, such as the Web. In such environments, clients request access to data located at remote servers. These requests are being processed at the server side and content is then delivered to the client. Latency, the time elapses since the user sends the request until the server’s response is accepted, is a major concern in many applications. The latency problem is exacerbated whenever network behavior is unpredictable or server processing time is considerable.

Minimizing latency is an obvious target of wide area environments and one of the more common solutions in practice is the use of client-side caching. A cache keeps copies of previously requested content and can respond to repeated client requests by using a local copy rather than requesting it from the origin

server, thus minimizing latency. Clearly, cached copies can only serve a client if the client deems them useful. In particular, stale copies are refreshed at the origin server.

The use of collaborative caching to enhance content delivery is based on the assumption that latency can be reduced if caches cooperate by exchanging content. Typically, the latency involved in accessing "near-by" caches is lower than accessing the origin server, and therefore a cache should join a coalition of caches, from which content can be downloaded. Although cache cooperation seems promising, analysis of Web proxy traces reveals that in many cases, it fails to provide significant improvements of hit ratio [19].

In this work, we aim at exploring the limitations of collaborative caching. We suggest a cooperative model for which cache content sharing show more promise. We explore recent developments in the research of replication management [3, 8], where a model of client tolerance towards stale data has been proposed. We propose a novel approach, based on this observation, in which clients can specify their tolerance using a simple-to-use method. The cache use this model and an update model at the server side to determine which of the following alternatives is most promising: a local copy, a copy from a cooperating cache, or a fresh copy from the origin server.

Throughout this work, we demonstrate the usefulness of our method for reusing dynamic content in a Web environment, generated from databases using templates. Dynamic content is created dynamically upon request, and may take orders of magnitude longer than retrieving static objects due to heavy database processing, and complex XML/HTML translations. For example, while high-performance Web servers can typically deliver thousands of static files per second, it is not uncommon for a script to consume over a second of CPU time when generating a single dynamic page [5], and when the server is loaded, users may experience very high latencies. While servers could greatly benefit from client-side caching for dynamically generated content, caches do not store dynamic content, and therefore cannot use it to respond efficiently to subsequent requests. We analyze this phenomenon and show that for a certain class of dynamic content, caching is beneficial. Such a model can be also useful for Content-Delivery-Networks (CDNs), in which cache cooperation can further reduce end-user latency [16].

We have built a simulation environment and tested our hypotheses. Our experiments reveal that using the proposed model makes it possible to meet user needs with reduced latency. We also show the benefit of cache cooperation reducing latency further. Specifically, we show that cache cooperation is particularly useful to users with high demands towards latency and consistency.

The specific contributions of this paper are as followed:

1. We analyze the failure of current cache cooperation and propose the terms under which such cooperation can become useful.
2. We present a decision making model for cooperative caching, in which a cache determines the appropriate method of processing a request, given user tolerance and content update models.

3. We show by detailed simulation the usefulness of cooperation in reducing latency, while maintaining a reasonable level of freshness under various settings.

The remainder of this paper is organized as follows. Section 2 provides background information and the formalism of our optimization problem. We next present an analysis of cache collaboration and qualitative criterions for successful cooperation (Section 3). Section 4 is devoted to presenting the user tolerance model. The decision making model is presented in Section 5, followed by empirical analysis in Section 6. We conclude with a review of related work (Section 7), and discussion of future work (Section 8).

2 Background

In this section we present a cache manager model, network characteristics, and the cost of latency and obsolescence that serve as a basis to our model, described in Section 4. We also introduce the formalism of our optimization problem. Throughout this section, we provide examples using the Web architecture and the HTTP protocol, and illustrate the complexity of contemporary content delivery using dynamic objects.

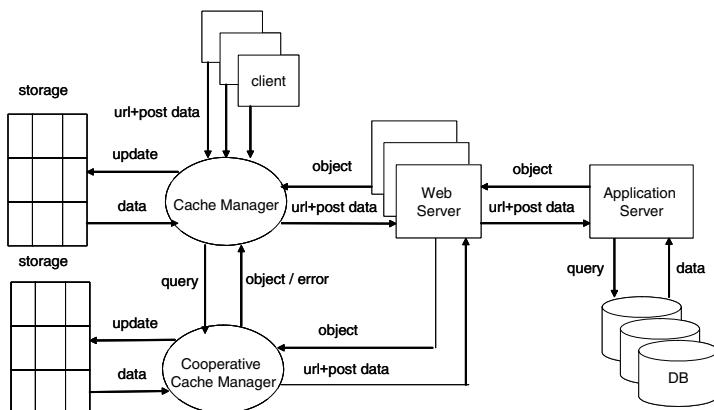


Fig. 1. Cache Manager Model

2.1 Cache Manager Model

When a user submits a request for an object, the browser forwards the request to a shared cache cache manager (see Fig. 1 for an illustration; the bottom left part of the figure illustrates the use of collaborative caches, to be discussed in Section 3). The cache manager searches for a valid object in its storage and returns it if found. Otherwise, the request is forwarded to the Web server. Upon

request, the Web server directs an application server to generate the content from one or more databases, and returns the constructed content to the cache manager. The cache manager then forwards the received content to the client's browser, occasionally leaving a copy at its storage for future use. Cached objects are stored with attributes like *Last Modification* and *Time-To-Live (TTL)*, set by the server, allowing the cache to decide upon copies validity.

2.2 Network Characteristics

When a cache handles a request, it dedicates a few tens of milliseconds to search for the object in its storage, loads it to memory, and transmits it back to the requestor. We refer to this time as *Internal Processing Delay*, and the latency involved with it is inevitable. The precise amount of time required to download a complete object from one cache to another depends on several factors including remote cache load, object size, network bandwidth, round-trip-time, and network configuration (*e.g.*, congestion control in the TCP protocol). Caches typically preserve the connection to the browser once opened, and response's packets are immediately sent through this connection, without waiting for the object to be completely downloaded. Therefore, the latency metric of interest is the *download latency*, which refers to the time from the moment the cache receives a request from a client until the first byte of the response returns. The *download latency* is the time needed to either generate a new object or locate an existing one, added to the *Round-Trip-Time (RTT)* between the two hosts involved. The *download latency* is not fixed over time, and might be affected by server loads, and network congestion. It is also worth noting that *RTT* between two hosts may change over time.

2.3 Latency, Obsolescence, and the Cache Optimization Problem

Web servers can attach a TTL value to Web objects, via HTTP directives such as 'expires', and 'max-age'. These directives provide the time up to which the object is considered valid. When Web servers do not set TTL value, caches use heuristics to decide when the next validation is required [14].

object staleness is defined as a binary term, where an object can either be fresh or stale. In [8], it was replaced with the term *object obsolescence*, where the object is gradually transformed from fresh to stale. During this process, objects may serve clients although they are obsolescent. Following [8], we propose a novel method for evaluating user tolerance towards object obsolescence in Section 4.

Updates of underlying data in many cases follow a systematic pattern [8, 15]. Discovering this pattern based on past behavior makes it possible to predict future updates. In [8], for example, a model for update estimation was proposed based on nonhomogeneous time-varying behavior (*e.g.*, more intensive activity during work hours, and less intensive activity after work). In our model we assume that update pattern of underlying data is known, and we make use of this knowledge to estimate object obsolescence. Based on [8], we define a combined cost of latency and obsolescence of an object $object_i$ as:

$$C_i = \alpha_{ik} L_i + (1 - \alpha_{ik}) O_i \quad (1)$$

where L_i is the download latency of $object_i$, and O_i is its obsolescence. C_i is the combined cost of latency and obsolescence, as evaluated by the cache for $user_k$. α_{ik} , $0 \leq \alpha_{ik} \leq 1$, stands for the ratio of importance $user_k$ relates to the latency involved with downloading $object_i$. At the one extreme, $user_k$ may specify $\alpha_{ik} = 0$, which is translated into a willingness to accept high latency. At the other extreme, $\alpha_{ik} = 1$ is translated into a preference for the smallest latency possible. Other values of α_{ik} allow the cache to utilize different copies of $object_i$ according to the latency associated with them. More advanced models, such as the one in WebPT [9] – where servers are categorized based on how noisy is the download data is – can be adopted as well.

Given $user_k$ request for $object_i$ and the availability of multiple copies of $object_i$ with varying obsolescence levels, the cache aims at minimizing C_i . We term this problem the *cache optimization problem*. We assume that object requests are independent, and therefore a global optimization problem (optimizing a set of user requests over a period of time) is separable in object requests. This assumption is reasonable, given existing cache architecture. We defer the discussion of more complex settings, such as cache prefetching [17] to a future research.

In order to enable the cache to estimate L_i and O_i , the server should cooperate by providing information regarding the state of the object, *e.g.*, creation timestamp of $object_i$ (t_i) and update rate of $object_i$ (λ_i). Update pattern can follow many distribution functions. Based on [6] we assume that Web objects are modified according to a Poisson process.

Based on these parameters, a cache can estimate the likelihood of an update to the underlying data as will be explained in Section 5.1. We also present in Section 5 an explicit representation of C_i that lends itself well towards cache cooperation and provides a model for solving the cache optimization problem using cache cooperation. First, however, we discuss cache cooperation in more detail.

3 Cache Cooperation

A cache may request content from other caches rather than from the origin server. There are two main conditions for such a cooperation to be useful. First, downloading from a cache should results in significantly lower latency. For example, dynamic content is stored at a cache as static information, which retrieval is much faster than its generation at the server. The second condition is that cooperating caches hold the necessary content to be delivered. Otherwise, such cooperation is no longer a valid alternative. Somewhat surprisingly, there are two extremes in which the second condition does not hold. The first, and more obvious extreme, is where the cache clients do not share the same interests and then the content in the caches will never overlap and cooperation becomes impossible. At the other extreme, whenever clients share extremely similar interests, caches tend to have similar content. In this scenario, a cache will not request for anything it can find in its own cache. If it so happens that a specific content cannot be found in one cache, it is likely that other caches will not have it

either. Experimenting with collaborative caching has yielded little benefit due to this extreme. As it turns out, the Zipf distribution of requests over the Web, in which some pages are extremely popular, resulted in very similar content across cooperating caches, rendering the cooperation useless [12].

To summarize, for cache cooperation to be productive, it should provide a significant reduction in latency and the overlap in client interest should be sufficiently large to allow exchange of content among caches, while still avoiding the extreme in which the overlap among caches makes cooperation useless.

In this work we present a setting, in which cache cooperation is beneficial. This setting involves rapid updates to content, which means that content, stored at a cache, becomes stale rather rapidly. Dynamic content is a typical example of rapidly changing content. Dynamic content is also costly to generate, and therefore latency is increased when accessing the origin server.

Another component that affects our model is user tolerance towards content obsolescence. The main observation here is that different users, while sharing interests regarding content, may exhibit dissimilar interests regarding object freshness. Therefore, while some prefer fresh objects even if latency is high, others favor low latency even at the cost of retrieving obsolescent objects.

In the next sections we present a user tolerance model and a decision process model for cache cooperation. We then provide empirical analysis to support our hypothesis.

4 User Tolerance Model

We now present our model of user tolerance. According to this model, users can quantify the importance they attach to reducing the latency inherent in the retrieval of content, and their willingness to risk the use of an obsolescent object. The first is modelled with α_{ik} , as discussed in Section 2.3. The second is modelled with p_{ik} , $0 \leq p_{ik} \leq 1$, which sets the threshold beyond which $user_k$ no longer considers $object_i$ sufficiently fresh as a function of the probability that it has been modified. p_{ik} fine-tunes the level of content obsolescence $user_k$ is willing to accept for $object_i$. The metaphor we use for p_{ik} is the risk one takes when choosing to leave home without an umbrella. This decision balances the chance of getting caught in the rain against the inconvenience of carrying an umbrella. One may choose to avoid any risk of getting wet, while another may choose to take an umbrella only if the odds of getting wet are pretty high. According to this "umbrella rule", a user is likely to risk using a possibly stale object, and enjoy the convenience of reduced latency, if the probability that an update has already occurred at the origin site is less than a pre-defined threshold (p_{ik}) set by the user. In other words, at the time the cache receives a request for $object_i$ from $user_k$, if the probability that the object has been modified exceeds p_{ik} then $object_i$ is considered obsolescent by $user_k$; otherwise, it is considered to be fresh. The probability of modification depends on the time the object was generated, and its update pattern, information that must be supplied by the server. The "umbrella rule" is illustrated in Fig. 2.

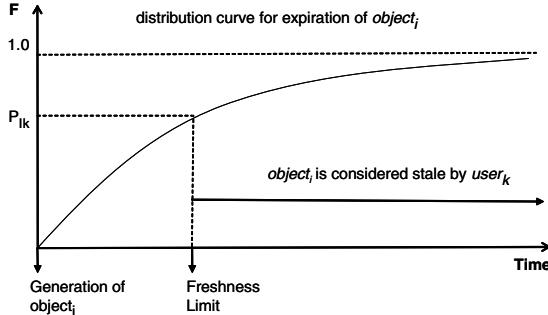


Fig. 2. User tolerance towards an object

Clearly, not all combinations of $\langle \alpha, p \rangle$ make sense. For example, $\langle 0.0, 1.0 \rangle$ constitutes an odd decision, since $\alpha = 0.0$ implies that the user will accept high latency, while $p = 1.0$ implies that the user will always consider the cached object to be fresh, a preference that would allow use of a cached copy to reduce latency. $\langle 1.0, 0.0 \rangle$ is equally unlikely. Still, our model permits any combination of $\langle \alpha, p \rangle$.

The ability to predict the likelihood that an object has been modified is needed to evaluate p . One such model was proposed in [8], in which updates in an origin server can be modelled as nonhomogeneous Poisson model. The model we propose in this work is independent of the specific update model. However, having such a model is a prerequisite for specifying user tolerance. It is worth noting that the update model is given in stochastic terms. Therefore, even if the update model is accurate, its probabilistic nature means that individual predictions can err. Therefore, whenever content consistency is critical, the level of tolerance towards stale content is expected to drop substantially, entailing frequent refreshes of the (possibly fresh) cached content. The decision made by the cache takes into account the $\langle \alpha_{ik}, p_{ik} \rangle$ values set by $user_k$, the update rate set by the server for $object_i$, as well as its estimated downloads latency and its creation timestamp. The exact formula is detailed in Section 5.1.

5 The Decision Making Process

In our model we use many factors, including user preferences, object characteristics, cache characteristics and environment. The symbols of these factors are described in Table 1. We next present the decision making process of a cache, when receiving a request for an object. Section 5.1 provides an explicit representation of the cost function. The model is discussed in Section 5.2.

5.1 Cost of Latency and Obsolescence

We aim at minimizing the combined cost of latency and obsolescence. Whenever a content request arrives at a cache, the cache has three alternatives it can

Table 1. Summary of symbols and their descriptions

Factor	Description
λ_i	update rate of $object_i$
t_i	generation timestamp of $object_i$
g_i	generation latency of $object_i$
u_i	popularity of $object_i$
α_{ik}	latency tolerance of $user_k$ towards $object_i$
p_{ik}	probability threshold of $user_k$ towards $object_i$ obsolescence
RTT	Round-Trip-Time between hosts
IPD	Internal-Processing-Delay of a cache
r	request inter-arrival-times
w	the weight of the last download latency

follow; It can retrieve the content from its local cache, request the content from a remote cache, or ask for a fresh copy from the origin server. The alternative that minimizes the total cost is then chosen. The validity of the first two alternatives depends on the availability of the content in the caches. We assume that accessing the remote server is always a valid alternative, ignoring possible downtime of servers. Extending the analysis to handle failures is beyond the scope of this paper.

The decision making process should compute the cost of each valid alternative. To do so, we now provide explicit instantiations of Eq. 1 for local cache, remote cache, and origin server content delivery. Eq. 1, based on [8], provides a generic form of a cost function, balancing latency and obsolescence. It does not take into account the difference in scaling of L_i and O_i . We therefore, modify it to the form of Eq. 2, adding F_n as a normalizing factor. For example, if L_i is measured in seconds and O_i is measured in minutes, we set F_n to be 1/60.

$$C_i = \alpha_{ik} L_i + F_n (1 - \alpha_{ik}) O_i \quad (2)$$

Each time a request for $object_i$ is sent by $user_k$ to a cache, the cache calculates the cost of valid alternatives and makes use of the alternative that minimizes the cost. In order to calculate the cost of each alternative, the cache must estimates L_i and O_i values. It is worth noting that both L_i and O_i cannot be determined deterministically. L_i stochastic nature stems from the noisy network environment and changing database loads, while O_i can only be accurately determined when accessing the server, something the cache attempts to avoid. L_i can be estimated by averaging the latency of previous downloads within a shifting window. O_i is much harder to estimate, since it must consider both the time elapsed since the generation of the object and its update rate. High update rate leads to a high estimation of O_i . It is important to note at this time that O_i is calculated as the time elapsed from the moment the user believes the object is no longer fresh until the moment of the request. The probability that $object_i$ had already been modified when a request arrives is calculated as $P\{x < t_{now} - t_i\}$, where t_i refers to the generation timestamp of $object_i$. As an example, assume that update inter-arrival times of $object_i$

are distributed exponentially with average λ_i . Then, given p_{ik} , the estimated period in which $object_i$ is considered sufficiently fresh for $user_k$ is calculated as:

$$P\{x < \Delta t\} = 1 - e^{-\lambda_i(\Delta t)} = p_{ik} \Rightarrow \Delta t = -\frac{\ln(1 - p_{ik})}{\lambda_i} \quad (3)$$

$user_k$ considers $object_i$ to be consistent if the request reaches the cache within $[t_i, t_i + \Delta t]$. During this interval, O_i is set to zero. After $t_i + \Delta t$ $user_k$ assumes that an update has occurred, and O_i is calculated as the time that has elapsed since $t_i + \Delta t$. For example, if $\lambda_i = 1/30$, and $object_i$ was generated five minutes before a request from $user_k$ arrives at the cache, then the odds that the object has been modified are $P\{x < 5\} \approx 0.15$. According to the umbrella rule described earlier, if $p_{ik} > 0.15$, then O_i is considered zero; otherwise it is considered obsolescent.

Cost of the locally cached alternative. For a locally cached copy, denoted $object_i^l$, L_i^l is taken to be zero as no latency is involved. O_i^l is calculated as the time elapsed since $t_i^l + \Delta t$ as follows:

$$O_i^l = \text{Max}\left(0, (\text{now} - t_i^l - \frac{\ln(1 - p_{ik})}{\lambda_i})\right) \quad (4)$$

Therefore, the cost of using the local object is given by:

$$C_i^l = (1 - \alpha_{ik}) \text{Max}\left(0, (\text{now} - t_i^l - \frac{\ln(1 - p_{ik})}{\lambda_i})\right) \quad (5)$$

For completeness sake, we assume that the C_i^l is set to infinity whenever this alternative is not valid.

Cost of the origin server alternative. For the origin copy, denoted $object_i^o$, O_i^o is taken to be zero, since a fresh object is evidently consistent. L_i^o is recalculated upon each download of $object_i^o$, based on a shifting window of past downloads, calculated as follows:

$$L_i^o = wL_i^o + (1 - w)g_i^o \quad (6)$$

g_i^o is the download time of $object_i$ origin copy, while w is used to tune the weights the cache puts on the past downloads versus the last one. The cost of using the origin object is given by:

$$C_i^o = \alpha_{ik} L_i^o \quad (7)$$

For completeness sake, we assume that the cost is set to infinity whenever this alternative is not valid. Such case can happen if the origin server is down or the object does not exist anymore.

Cost of the remote cache alternative. For the remote copy (located at some collaborative cache), denoted $object_i^r$, both L_i^r and O_i^r should be evaluated. L_i^r is calculated as follows:

$$L_i^r = RTT + IPD \quad (8)$$

RTT refers to the Round-Trip-Time between the caches, and *IPD* refers to Internal-Processing-Delay inside a cache whenever a request is processed. As *RTT* and *IPD* can change over time, each cache should measure and update these values from time to time. O_i^r is calculated as follows:

$$O_i^r = \text{Max} \left(0, (\text{now} - t_i^r - \frac{\ln(1 - p_{ik})}{\lambda_i}) \right) \quad (9)$$

Therefore, the cost of using the remote object is given by:

$$C_i^r = \alpha_{ik}(\text{IPD} + \text{RTT}) + (1 - \alpha_{ik}) \text{Max} \left(0, (\text{now} - t_i^r - \frac{\ln(1 - p_{ik})}{\lambda_i}) \right) \quad (10)$$

For completeness sake, we assume that the cost is set to infinity whenever this alternative is not valid. Clearly, if multiple remote copies are available, the cost should be evaluated for each copy separately.

After calculating the cost of all available copies, the cache then chooses the copy with the lowest cost to fulfil the request at hand.

5.2 Discussion

Equipped with the user tolerance model presented in Section 5.1, a cache can choose among various copies of a requested object, according to user preferences and cost estimations. Assuming each cached object has an update pattern, it is easy to estimate the probability of its updates, and estimate its validity against user preferences. The cache considers user preferences regarding latency (α_{ik}), and user tolerance towards obsolescence (p_{ik}), when choosing among the available alternatives. High values of p_{ik} indicate high tolerance towards obsolescence, thus encouraging use of a cached copy.

Existing decision making processes treat all users equally, not allowing them to express any preferences, and use a cached object as long as its TTL has not expired. Only when the object has expired the cache may request an object from a collaborative cache, or from the origin server. Thus, users may suffer from stale content although they expect fresh information (since TTL has not expired yet), and they cannot benefit from fresher content that resides at a neighboring cache as long as TTL is valid. On the other hand, users may suffer from high latency if TTL is short. With the proposed decision making process, the cache has more flexibility and it can fine-tune the decision for each user according to her needs.

We believe that cache cooperation in the Web environment did not yet fulfill its potential. With frequent updates of objects and diverse preferences of users towards obsolescence and latency the cooperation can be more effective, and the cache performances will be significantly improved.

A final comment regarding the cost of cooperation is warranted at this time. When using a cooperation schema, one must be aware of the overhead involved, and consider it while estimating its benefits. For example, in the directory-based implementations, caches share information regarding their storage content. A notification about changes occurring at each cache is sent to all collaborators,

and might burden the cache with many such notifications, not to mention the overhead of managing a huge directory. However, in the directory-based cooperation case the cost of directory management can be amortized over all requests, and in this work we assume that the cost of the cooperation mechanism is acceptable by the cooperating caches. For the sake of simplicity we ignore these overhead costs, and concentrate on the potential benefits of cache cooperation regardless of specific implementation. Overhead considerations are left open for future work.

6 Empirical Analysis

We have built a simulation environment and produced synthetic data to examine the impact of different caching strategies on the usefulness of cache cooperation and its impact on cache performance (latency) and content obsolescence. We have examined the effect of numerous parameters on cache performance, including number of cooperating caches, user preferences, and object update rate.

Caching strategies are detailed in Section 6.1. The simulation environment and its parameters are described in Section 6.2. Finally, we present the evaluation metrics (Section 6.3), the experimental results (Section 6.4) and an empirical analysis of our results (Section 6.5).

6.1 Caching Strategies

We examined two strategy categories, namely TTL-based and Profile-based. As for the TTL-based strategy we implemented two variations:

Flat TTL: Servers set a flat TTL for all objects, within which the object is considered fresh, and is used to serve requests. We examined various TTL values ranging from 0 – 60 minutes.

MT-TTL: Server sets individual TTL for each object based on the mean-time (MT) between successive updates. We examined various TTL values ranging from 25 – 100% of the mean-time between successive updates. It is worth noting that MT-TTL is an example of a strategy that uses history (unlike TTL that uses only recent modification time), yet does not consider user preferences.

The Profile-based strategy is based on the decision making process of Section 5. We have used the following three preference profiles (to all objects):

High-Recency: $\langle \alpha = 0.05, p = 0.05 \rangle$. This profile sets 5% importance to low latency, and a 0.5 threshold probability for freshness.

Low-Latency: $\langle \alpha = 0.95, p = 0.95 \rangle$. This profile sets 95% importance to low latency, and a 0.95 threshold probability for freshness.

Random: This profile sets random α and p values, set independently between 0 – 1.

6.2 Simulation Settings

Update rates of the objects (λ_i), and request load on the caches (r) were modelled using exponential distribution. While r was set to 125 ms for all the caches, λ_i was randomly chosen from the range 1 – 30 minutes, for each object. Object download time was also randomly chosen from the range 0.5 – 5 seconds, and used as fixed download time. Object popularity (u_i), was modelled with Zipf-like distribution, where the popularity of the i -th most popular object is proportional to $1/i^\beta$, for some constant β between 0 and 1. Analysis performed on a few cache traces reported β between 0.64 – 0.83 [2], Therefore, we set $\beta = 0.75$. We also examined our model with objects that exhibit identical popularity across all caches, and with dissimilar popularity as well. It is important to note that the values assigned to u_i , λ_i , and g_i were set independently. As for the weight of the last download we used $w = 0.3$ to emphasize the relevance of near-past downloads.

RTT between the caches, and between cache and server were set to fixed values of 50ms and 250ms, respectively. We also set *IPD* to be fixed 20ms for all caches.

6.3 Metrics

We demonstrate our model on a collaboration schema assuming all caches share information about their objects. To perform the experiments, we implemented a simulation environment in Java. The code mimics the behavior of a cache, extending its capability to handle dynamic objects. We report on the following cache metrics:

Average latency. The average duration between receipt of a request by the local cache and the time it receives the first byte of the response.

Hit ratio. The ratio of requests served by the **local** cache from its storage, or from a **remote** collaborative cache.

Stale delivery ratio. The ratio of requests that were served stale from the cache. Note that this parameter measures the actual staleness of an object, and is independent of user tolerance towards obsolescence.

6.4 Experimental Results

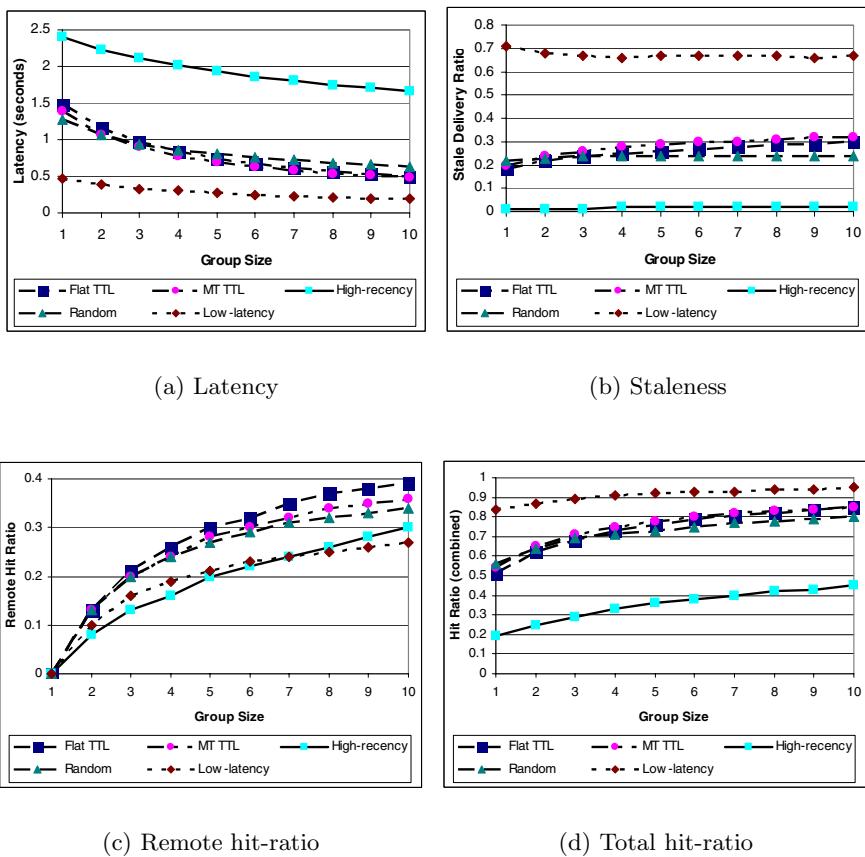
Out of the 10 hours of the simulation, the first two hours were dedicated to warm up the caches. The server offers 10,000 objects, and each cache handles approximately 230,000 requests during the simulation. To avoid biases, we assumed unlimited cache capacity.

We first examine how a single cache performs, depending on the strategy chosen. The simulation results are summarized in Table 2.

Clearly, the use of the Flat-TTL strategy with zero TTL led to the worst latency, since caching is not possible, and all requests are served by the origin server. With this strategy, users get fresh content, but the latency is high. Users which prefer small low latency, and willing to accept potential content obsolescence, have no choice but to wait for the origin server response. As we amplify

Table 2. Cache performances with a single cache

Strategy	Value	Average Latency	Hit Ratio	Stale Ratio
Flat TTL	1 minutes	2.37	0.21	0.02
	10 minutes	1.49	0.51	0.18
	60 minutes	0.65	0.79	0.61
MT TTL	0.25MT	1.98	0.34	0.04
	0.5 MT	1.69	0.44	0.1
	1 MT	1.38	0.54	0.2
Profile	High-Recency	2.41	0.19	0.01
	Random	1.28	0.56	0.22
	Low-Latency	0.46	0.84	0.71

**Fig. 3.** Cooperation and its impact on cache performances

the TTL value, the hit ratio grows and the latency is reduced accordingly, as shown in Table 2. Yet, the ratio of stale services increases proportionally.

When MT-TTL is used, TTL is set to individual objects, based on their update rate, thus results are expected to exhibit better performances than flat

TTL, which treats all objects uniformly, regardless of their update rate. As shown in Table 2, the MT-TTL with 100% mean-time exhibits higher hit ratio than flat TTL with 10 or even 30 minutes and accordingly reduced latency. Yet, since the mean-time is merely an estimation, the ratio of stale services is high. When we tune the TTL to be 50% or 25% of the mean-time, we indeed achieve lower rates of stale content ratio, but the hit ratio drops, and the latency increases.

The Profile strategy with High-Recency preferences achieved better latency than those achieved in Flat-TTL with zero value and similar to Flat-TTL with 1 minute. Although users preferred up-to-date content, the cache served almost fifth of the requests from its local storage, of which only 1% were stale. With the Low-Latency preferences we achieved the most dramatic improvement in the latency metric, with 84% hit ratio. As expected, the stale ratio increased in the Low-Latency case to 71%. The increased proportion of stale delivery as the profile become more tolerant to obsolescence stems from the increasing importance users place on low latency. Results in the Random case fell in between the other profile strategies, and resembled the MT-TTL strategy results achieved for 1MT. Yet, in the random case, as opposed to MT-TTL, users can optimize their choice.

After exploring the behavior of a single cache, we repeated the simulation with identical settings, except that now we configured varying number of cooperating caches and measured how cooperation assists caches with increased hit ratios, and reduced latency. The load on each cache remains the same. For the sake of brevity we present in Fig. 3 only the results attained for Flat-TTL with 1 minute, MT-TTL with 1MT, and the profiles strategies.

Figure 3(a) presents average latency, as affected by the number of cooperating caches. It reveals that all strategies benefit from cooperation significantly. Figure 3(b) presents the rate of stale deliveries and it shows that all the Profile-based strategies keep the stale delivery rate stabilized while TTL strategies suffer from higher rates of stale deliveries as the group size increases.

Figure 3(b) also shows that communities that prefer High-Recency cuts approximately one third of the latency due to cooperation almost without loosing freshness, while the common strategy of zero TTL does not benefit from cooperation at all. Communities preferring low latency also benefit from cooperation while ratio of stale deliveries remains steady. With TTL-based strategies cooperation increases the ratio of stale deliveries, probably because a rise in the number of cooperative caches makes available more dissimilar copies of each object that become stale but are considered fresh enough for users. With more requests served by these cashes, fewer are forwarded to the server, and so fewer fresh objects are served.

Figure 3(c) reveals that cooperation contributes a higher proportion of remote hits as group size increases. Figure 3(d) presents the combined local and remote hits. It can be observed that cooperation contributes to higher hit ratios, thus smaller latency. It can also evident that the marginal degree of improvement falls as the group grows in size.

Another set of tests were designed to explore how different update rates affect the remote hits and the benefit of cooperation. We tested all strategies, but for

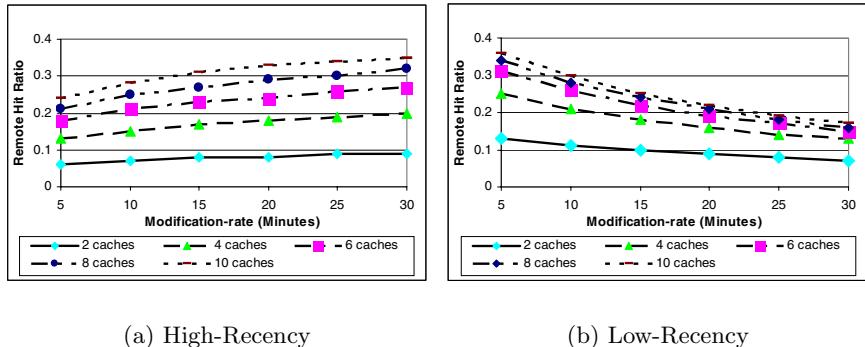


Fig. 4. Cooperation and update rates

the sake of clarity we present here only the results attained for low latency and high recency. We run the simulation with varying number of cooperating caches and with varying update rates.

Figure 4 presents the remote hit ratios attained for High-Recency and Low-Latency profiles. Figure 4(a) reveals that cooperation become more useful as the mean time between successive updates grows. Somewhat surprisingly, Fig. 4(b) reveals that with low-recency profile, as the mean-time between updates grows, cooperation decreases. This may be attributed to the use of a locally cached object. In the High-Recency case an object becomes costly rather quickly, and cooperating cache copies assist in reducing latency. As the average time between updates grows, the chances to find a fresher object in the cooperating caches grows, and thus cooperation increases. With Low-Latency profile, a cache can use the local copy many times, but when it becomes too obsolescent, most chances are that neighboring copies are identical to the costly copy, since all caches serve communities with low latency, which also used their local copy many times, thus cooperation become less useful. As for the other strategies, the Flat-TTL is not affected by the update pattern, since its TTL value is set arbitrarily, but the MT-TTL and the Random profile tend to behave like Low-Latency, with usefulness of cooperation dropping much faster than that of High-Recency.

6.5 Discussion

The profile strategy is empirically proven to be advantageous over the TTL-based strategies in all respects. Caches can adapt their performance to the needs of their users; and with a user preference and object evolution model, decisions about which copy of the object to serve are more flexible than with other strategies. Clients requiring fresh content will get a copy from the origin server, regardless of the latency involved. Clients requiring a fast response may get a locally cached copy, however obsolescent it may be. Adapting cache decisions to user preferences and object update rates makes it possible to increase the number of objects served by local cached copy or a neighboring copy, and thereby

reduce average latency, while still satisfying users' obsolescence specifications. Another outcome of the simulation is that the Profile strategy with homogenous preferences benefits collaboration with other caches while keeping ratio of stale deliveries stabilized, and when high recency is required the cache can provide it fresh and faster than TTL-based strategies.

When objects gets updated often at the server, cooperation is even more useful for communities with low latency preferences, while less frequent updates increase cooperation usefulness for communities with high recency preferences.

To conclude, strategies that ignore user preferences and consider only TTL set by the server leads to lower performance, in comparison with profiles that consider user preferences, especially when cooperation with other caches is possible. TTL-based strategies do not cater to specific user needs, and so cannot adapt to different preferences. Profile strategies are thus best suited to adapting to user varying needs while taking into account object update rates, and they therefore provide the best performance, and cooperation attain better results under such strategies.

7 Related Work

In [3] it was suggested that users supply their preferences regarding target-latency and target-recency of Web objects, along with a weight, setting their relative importance. The cache would then score its local copy against the origin server's copy and serve the most appropriate version. The cache would estimate recency and latency based on log information of past downloads. Our model suggests a simpler and more intuitive method to model user preferences, and examines how Profile-based cache strategy benefits cache cooperation. In [10], a generic cost function introduced, to consider costs of latency and recency, and in our work we extended it to assist cache with decision making as discussed in this paper.

The performances of cache cooperation and its characteristics have been studied intensively in recent years. Cooperation schema in the Web environment was introduced in [18, 1] where upon a miss, a query is sent to all collaborators before querying the server. But cache cooperation benefits were shown to be limited, and many studies explored these limitations. For example, [2, 4, 19] have found that hit ratio for proxy caches grows rapidly when clients are added, but flattens after a certain point. In general, hit ratio grows in a log-like fashion with the number of clients and requests. In [12] the effectiveness of cooperative caching was reported to decrease as the skew in object popularity increases. Higher skew means that only a small number of objects are most frequently accessed reducing the benefit of larger caches, and therefore of cooperation. Other aspects of cache cooperation were also explored including coordinated placement and replacement policies [10], and the overhead of cooperation mechanisms and their affect on cooperation [7]. In [10], coordinated placement and replacement was shown to significantly improve performances in cache hierarchies. In [7], it was reported that cooperation is viable for distributed cooperation mechanisms, and

not viable in hierarchies. These observations led to the conclusion that cooperation effectiveness depends on the size of the working set and the cache capacity, and therefore it is not always beneficial. Web traces of cooperative caches were analyzed in [11], concluding that hit ratios were improved via cooperation, but the improvements rate varies widely between 0.7% and 28.9%. In [13] a distance sensitive model was developed to use cooperation only when it is beneficial, and turn it on and off dynamically.

Our work differs from other works in that we have taken a different angle on cooperation, under varying user preferences regarding latency and recency affects, and further explore how update rates affect cooperation usefulness.

8 Conclusions and Future Work

In this paper we examine the usefulness of cache cooperation, given user profiles regarding latency and data obsolescence. Users provide their tolerance towards obsolescent content and the emphasis they put on rapid response time. We then proposed the use of cached content at either local or remote caches, even in cases where content may no longer be fresh yet still within a user tolerance boundaries. Our empirical analysis shows that by employing user preferences, caches can share content and tune their performance to suit user needs. We further show empirically that cooperation with other caches is beneficial, in particular to High-Recency profiles, in which users demand low obsolescence as well as low latency. While we have proven our model to be useful on syntactic data, we plan to validate our model on real data set, consisting of requests made to 1998 World Cup Web site¹.

Several topics are left open for future research. While cache cooperation carries a promise of big benefit to users, it also has to consider the overhead of cooperation. We plan on investigating the various methods of cache cooperation, analyze the cooperation overhead and develop a model for balancing user benefit with cache resource utilization. As another topic, we intend to investigate the impact of cache size and various replacement policies on cooperation. Finally, we plan to explore how caches can choose collaborators wisely and efficiently.

References

1. C. M. Bowman, P.B. Danzig, D.R. Hardy, U. Manber, and M.F. Schwartz. The Harvest information discovery and access system. *Computer Networks and ISDN Systems*, 28(1–2):119–125, 1995.
2. L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of IEEE Infocom*, pages 126–134, 1999.
3. L. Bright and L. Raschid. Using latency-recency profiles for data delivery on the Web. In *Proceedings of the International conference on very Large Data Bases (VLDB)*, pages 550–561, Hong Kong, China, August 2002.

¹ The data was taken from <http://ita.ee.lbl.gov/html/contrib/worldcup.html>

4. P. Cao and S. Irani. Cost-aware www proxy caching algorithms. *Proc. USENIX Symposium on Internet Technologies and Systems*, 1997.
5. J. Challenger, A. Iyengar, P. Dantzig, D. Dias, and N. Mills. Engineering highly accessed Web sites for performance. *Lecture Notes in Computer Science*, 2016:247–265, 2001.
6. J. Cho and H. Garcia-Molina. Estimating frequency of change. *ACM Transactions on Internet Technology (TOIT)*, 3(3):256–290, 2003.
7. S.G. Dykes and K.A. Robbins. A viability analysis of cooperative proxy caching. In *Proceedings of IEEE Infocom*, pages 1205–1214, 2001.
8. A. Gal and J. Eckstein. Managing periodically updated data in relational databases: a stochastic modeling approach. *Journal of the ACM*, 48(6):1141–1183, 2001.
9. J.R. Gruser, L. Raschid, V. Zadorozhny, and T. Zhan. Learning response time for WebSources using query feedback and application in query optimization. *VLDB Journal*, 9(1):18–37, 2000.
10. M.R. Korupolu and M. Dahlin. Coordinated placement and replacement for large-scale distributed caches. Technical Report CS-TR-98-30, 1, 1999.
11. P. Krishnam and B. Sugla. Utility of co-operating Web proxy caches. *Computer Networks and ISDN Systems*, 30(1–7):195–203, 1998.
12. K.W. Lee, K. Amiri, S. Sahu, and C. Venkatramani. On the sensitivity of cooperative caching performance to workload and network characteristics. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 268–269, New York, NY, USA, 2002. ACM Press.
13. M. Rabinovich, J. Chase, and S. Gadde. Not all hits are created equal: cooperative proxy caching over a wide-area network. *Computer Networks and ISDN Systems*, 30(22–23):2253–2259, 1998.
14. M. Rabinovich and O. Spatschek. *Web caching and replication*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
15. A.I.T. Rowstron, N. Lawrence, and C.M. Bishop. Probabilistic modelling of replica divergence. pages 55–60, 2001.
16. A. Vakali and G. Pallis. Content delivery networks: Status and trends. *IEEE Internet Computing*, 7(6):68–74, 2003.
17. Z. Wang and J. Crowcroft. Prefetching in world wide web, 1996.
18. D. Wessels and K. Claffy. ICP and the Squid Web cache. *IEEE Journal on Selected Areas in Communication*, 16(3):345–357, 1998.
19. A. Wolman, G.M. Voelker, N.S., N. Cardwell, A.R. Karlin, and H.M. Levy. On the scale and performance of cooperative web proxy caching. In *Symposium on Operating Systems Principles*, pages 16–31, 1999.

A Data Stream Publish/Subscribe Architecture with Self-adapting Queries

Alasdair J.G. Gray and Werner Nutt

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh, EH14 4AS, UK

Abstract. In data stream applications, streams typically arise from a geographically distributed collection of producers and may be queried by consumers, which may be distributed as well. In such a setting, a query can be seen as a subscription asking to be informed of all tuples that satisfy a specific condition. We propose to support the publishing and querying of distributed data streams by a *publish/subscribe architecture*.

To enable such a system to scale to a large number of producers and consumers requires the introduction of *republishers* which collect together data streams and make the merged stream available. If republishers consume from other republishers, a hierarchy of republishers results.

We present a formalism that allows distributed data streams, published by independent stream producers, to be integrated as views on a mediated schema. We use the formalism to develop methods to adapt query plans to changes in the set of available data streams and allow consumers to dynamically change which streams they subscribe to.

1 Introduction

Data streams usually record measurements that originate from sensors which are installed at multiple locations, or which may even be moving, e.g. in road traffic monitoring. Similarly, users interested in the streams are often distributed as well. We propose to understand the management of such streams as a data integration task [9].

As opposed to the *one-time* queries posed over a database, which are interested in receiving a set of answers against the current set of data, queries over data streams are *continuous*. One can distinguish three main ways in which such queries are used. The first is to direct data to a database where it is archived to be queried later on. Another is to feed monitoring data produced by sensors into databases that maintain a cache of current data and reflect the current state of the entities being monitored. Finally, continuous queries may be set up to detect classes of events. Such queries scan a stream for data values that exceed some threshold or that form specific patterns. Often, considerable leverage can already be gained if such tasks are realised for relatively weak query languages [4,12].

In the present paper, we build upon a data integration approach to publishing and querying data streams which has been partially implemented in the R-GMA Grid information and monitoring system [3,4] which is now being deployed in

major production Grids, e.g. LCG. The approach follows the proposal for a Grid Monitoring Architecture (GMA) [13] made by the Global Grid Forum. Although these ideas have been developed first in the context of Grid monitoring, they are far more general and can be applied to other scenarios where queries are posed over distributed streams.

In this approach, sources of stream data are called *producers* while entities posing queries are called *consumers*. Producers and consumers interact with each other according to a publish/subscribe protocol. A registry service allows producers to advertise what kind of data they can supply, and consumers to register what kind of data they request. Both advertisements and requests take the form of queries over a set of relations which together make up a *global schema*. The rôle of the mediator [14] is shared by the registry service, which finds suitable producers for a query, and the consumer, which constructs a collection of queries over these producers and executes them.

Since components can only maintain a limited number of connections, we introduce *republishers* to enable such a system to scale to a large number of producers and consumers, as necessary for a Grid. A republisher poses a query over the global schema, and makes the answer stream available for other queries. As such, these republishers can be used to create a *hierarchy*. At the bottom of such a hierarchy would be the producers. These would feed into the republishers with the most specific queries. In turn, these republishers would be used by more general republishers. The need for something like a republisher hierarchy has been identified in other Grid monitoring systems. For example, the widely used MDS2 system [5] has an information collection service called a GIIS which can be formed into a hierarchy. In such a hierarchy, the bottom level GIISs collect information directly from the data sources. These then feed information into GIISs at the higher levels of the structure.

The formation of a hierarchy is useful as it (i) collects the “trickles” of information produced by sensors and merges them into data streams, (ii) reduces the number of connections that any one component needs to maintain, e.g. a consumer need not contact every producer of relevant information if there exists a republisher which already merges this data together, and (iii) can be used as a cache of either latest-state information or historical data by storing the data streams into a suitable database.

Whilst the presence of republishers allows for more efficient ways to answer a consumer query, they also introduce complications. First, with republishers, redundancy arises amongst data, since a piece of data that has been published by a producer may also be made available by one or more republishers. Thus, when answering a query, a choice has to be made as to where to retrieve which data.

Secondly, continuous queries are issued at some point in time and continue to return answers until explicitly ended. This means that producers or republishers may be added to, or removed from, the hierarchy during the lifetime of a query and techniques are needed to adjust consumer query plans to such changes. Related to this is the issue of maintaining the hierarchy. The formation

of hierarchies in the MDS2 system, for instance, is a manual process. Thus, if a GIIS should become unavailable there is no mechanism by which the system can adapt to overcome this.

Building upon previous techniques for planning consumer queries in the presence of stream republishers [3,8], the main contributions of this paper are to (i) formalise the techniques for adapting queries when the set of data sources changes, and (ii) develop suitable mechanisms to enable consumers to switch between data sources.

The rest of this paper is structured as follows. In Section 2 we present the approach developed for publishing and querying distributed data streams. Section 3 then considers how to maintain these query plans. In Section 4 we identify desirable properties for a hierarchy of republishers and in Section 5 we develop methods for switching between data sources. We present related work in Section 6 and our conclusions in Section 7.

2 Publishing and Querying Relational Data Streams

A stream publish/subscribe network consists of three kinds of components: producers, consumers, and republishers. A stream producer generates and publishes a stream of tuples; a consumer poses a query, by which it requests a stream consisting of all tuples that satisfy the query; a republisher, similar to a consumer, poses a query but also publishes the answers to that query. In order for the components to communicate with each other, there is a registry service, which knows about all components existing at any given point in time.

This section summarises the approach for publishing and querying data streams developed in [3]. In this paper, we consider only continuous queries that are selections on a relation. This is similar to the class of continuous queries currently supported by the R-GMA system, where continuous queries are used to control the flow of data and more complex queries are posed over the caches held by the republishers.

2.1 Publishing Relational Data Streams

We assume that there is a *global relational schema* against which consumers and republishers pose their queries. The attributes of a relation in the global schema are split into three parts: key attributes, measurement attributes, and a timestamp attribute. As an example, taken from a grid monitoring application, consider the relation `ntp` (“network throughput”) with the schema

$$\text{ntp}(\underline{\text{from}}, \underline{\text{to}}, \underline{\text{tool}}, \underline{\text{psize}}, \text{latency}, \text{timestamp}),$$

which records the time it took, according to some particular tool, to transport packets of a specific size from one node to another. The underlined attributes make up the primary key of `ntp`, while `latency` is the measurement attribute.

Producers inform the registry about the data they publish by registering a selection query $\sigma_D(r)$ or *view* with the registry service. This view is built up

using the usual operators in such queries, such as equalities and comparisons, and combining them by boolean connectives. If the relation r does not yet exist in the global schema, it is added to it. For example, a producer S_1 that publishes a data stream consisting of network latency measurements, originating at Heriot-Watt University, made with the `UDPmon` tool would register the view $\sigma_{\text{from}=\text{'hw'} \wedge \text{tool}=\text{'udpmon'}}(\text{ntp})$.

The meaning of such a registration is that the producer promises to publish only tuples that satisfy the view. We say that such a producer *publishes for* r . Thus, the producer S_1 promises to publish tuples which record the latency of packets being sent from Heriot-Watt University measured with the `UDPmon` tool. The view registered by a producer is only a sound, but not a complete description of its data. It is possible, therefore, that different producers register identical or overlapping views.

For our discussion, we adopt a sequence-based model of streams. A stream may be infinite or end after a finite time. We capture this by defining a stream s for relation r as a partial function from the natural numbers \mathbb{N} to the set T_r of all tuples satisfying the schema of r ,

$$s: \mathbb{N} \hookrightarrow T_r,$$

such that, if $s(n)$ is defined for some $n \in \mathbb{N}$, the tuple $s(m)$ is defined for all $m < n$. Thus, $s(n)$ denotes the n^{th} tuple of s . We assume that each producer publishes a stream of tuples satisfying its descriptive view.

To specify further assumptions, we need two shorthands for the subtuples of $s(n)$. We write $s^\kappa(n)$ for the values of the key attributes and $s^\tau(n)$ for the timestamp. We say that a stream s_1 is a *substream* of s_2 if s_1 can be obtained from s_2 by deleting zero or more tuples from s_2 . A *channel* of s is a maximal substream whose tuples agree on the key attributes of s . For a tuple t occurring in s , where t^κ is the subtuple of t that contains the values of the key attributes, the substream of s consisting of the tuples with $s^\kappa(n) = t^\kappa$ is the *channel of* t .

We require that the conditions D in producer views only restrict the key attributes. Thus, the view restricts the channels of a producer, but not the possible measurements.

We say a stream s is *duplicate free* if for all m, n with $m \neq n$ we have that $s(m) \neq s(n)$. A stream s is *weakly ordered* if for all m, n with $s^\kappa(m) = s^\kappa(n)$ and $m < n$ we have that $s^\tau(m) < s^\tau(n)$. This means that in every channel of s , tuples appear in the order of their timestamps. Two streams s_1 and s_2 are *channel disjoint* if for all m, n we have that $s_1^\kappa(m) \neq s_2^\kappa(n)$, that is, if s_1 and s_2 have no channels in common.

In addition to satisfying the descriptive views, we require that the streams published by producers in a network are duplicate free and weakly ordered. Many complex queries over streams, e.g. aggregate queries, are defined for streams with some degree of order. Since it is impractical in a distributed setting to achieve that streams are completely ordered w.r.t. their timestamps, we only guarantee that each channel is ordered. Distinct producers must publish streams that are channel disjoint, although their views may overlap.

2.2 Queries and Query Plans

Consumers and republishers pose arbitrary selection queries over the global schema, which have the form $\sigma_C(r)$. For example, a consumer interested in all latency measurements for packets of size at least 1024 bytes that have been sent from Heriot-Watt University would pose the query $q_1 := \sigma_{\text{from}=\text{hw} \wedge \text{psize} \geq 1024}(\text{ntp})$. Similarly, a consumer interested in links with a low latency measured by the ping tool would pose the query $q_2 := \sigma_{\text{tool}=\text{ping} \wedge \text{latency} \leq 10.0}(\text{ntp})$. The only assumption we make is that satisfiability and entailment of such conditions is decidable. Both consumers and republishers register their queries with the registry service. The query of a republisher also serves as a complete description of the data stream that it publishes, i.e. its view.

We will now consider how to answer a consumer selection query $q = \sigma_C(r)$. Since there is no well-defined stream r , we have to define what an answer to such a query should be. An *answer stream* for q is any duplicate free and weakly ordered stream that consists of those tuples satisfying C that occur in streams of producers that publish for r . Note that, according to this definition, there can be infinitely many different answer streams for a query q . Any two answer streams consist of the same tuples, but differ regarding the order in which they appear.

A query referring to a relation r in the global schema is called a *global query*. A global query cannot be answered by accessing r directly, since r stands only for a virtual stream. Instead, a global query has to be translated into a *local query*, which accesses producers and republishers, which publish physical streams. We refer jointly to producers and republishers as *publishers*, denoted as P , and to the set of all publishers as a *publisher configuration*, denoted as \mathcal{P} .

To answer global selection queries it is sufficient to consider local queries Q that are multiset unions of selections over publishers P_1, \dots, P_m , written

$$Q = \sigma_{C_1}(P_1) \uplus \dots \uplus \sigma_{C_m}(P_m).$$

An *answer* to Q is any stream that is obtained by selecting, for each $i = 1, \dots, m$, the tuples satisfying C_i from the stream published by P_i , and then merging the resulting streams.

A local query Q is a *plan* for a global query q if every answer stream for Q is also an answer stream for q . Thus, Q is a plan for q if and only if all possible answer streams for Q have four properties. They have to be *sound* and *complete* w.r.t. q , that is, they have to contain *only* tuples satisfying q and they have to contain *all* such tuples. Moreover, they have to be *duplicate free* and *weakly ordered*. We say that a local query has any of these four properties if its answer stream always has this property.

Let us consider a publisher configuration \mathcal{P}_1 consisting of five producers measuring the network latency from three sites, Heriot-Watt University (**hw**), Rutherford Appleton Laboratory (**ral**), and Agia Napa (**an**), using two distinct tools. The views registered by these producers are

$$\begin{aligned} S_1 &:= \sigma_{\text{from}=\text{hw} \wedge \text{tool}=\text{udpmon}}(\text{ntp}), \quad S_2 := \sigma_{\text{from}=\text{hw} \wedge \text{tool}=\text{ping}}(\text{ntp}), \\ S_3 &:= \sigma_{\text{from}=\text{ral} \wedge \text{tool}=\text{ping}}(\text{ntp}), \quad S_4 := \sigma_{\text{from}=\text{ral} \wedge \text{tool}=\text{udpmon}}(\text{ntp}), \\ S_5 &:= \sigma_{\text{from}=\text{an} \wedge \text{tool}=\text{ping}}(\text{ntp}). \end{aligned}$$

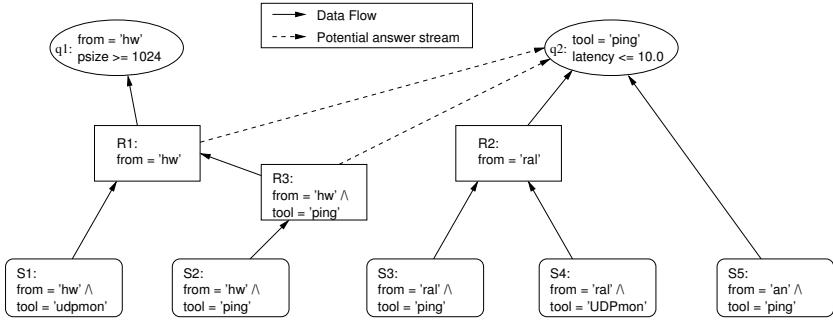


Fig. 1. The consumer query q_1 being posed at the publisher configuration \mathcal{P}_1

The publisher configuration also contains three republishers. The **hw** and **ral** sites each have a republisher which collects together all the latencies measured at that site. These would register the views $R_1 := \sigma_{\text{from}=\text{'hw'}}(\text{ntp})$, and $R_2 := \sigma_{\text{from}=\text{'ral'}}(\text{ntp})$ respectively. There is also a republisher which republishes all ping measurements made from **hw**. This republisher would register the view $R_3 := \sigma_{\text{from}=\text{'hw'} \wedge \text{tool}=\text{'ping'}}(\text{ntp})$.

Now consider the consumer query q_1 being posed at \mathcal{P}_1 . A suitable query plan for q_1 would be

$$Q_1 = \sigma_{\text{from}=\text{'hw'} \wedge \text{psize} >= 1024}(R_1). \quad (1)$$

Fig. 1 illustrates the flow of data between the publishers in \mathcal{P}_1 . We will discuss the possible query plans for q_2 in the next section.

2.3 Computing Consumer Query Plans

We assume from now on that in global queries and descriptive views the conditions on key and measurement attributes are decoupled, that is, every condition C can be equivalently written as $C^\kappa \wedge C^\mu$, where C^κ involves only key attributes and C^μ involves only non-key attributes. In practice, this assumption will usually be satisfied. The general case requires quantifier elimination techniques.

We first consider consumer queries. To begin with, we single out the publishers that can contribute to a query plan. We say that a publisher P with view $\sigma_D(r)$, where $D = D^\kappa \wedge D^\mu$, is *relevant* for a query $q = \sigma_C(r)$ with $C = C^\kappa \wedge C^\mu$ if it has the following two properties:

1. $C \wedge D$ is satisfiable (Consistency);
2. $C^\mu \models D^\mu$ (Measurement Entailment).

The first property states that P can potentially contribute values for *some* channels requested by q . The second one states that for those channels *all* measurements requested by q are offered by P . This ensures that all the plans computed generate answer streams which are weakly ordered. Note that if P is a producer, measurement entailment always holds because the conditions in the view describing P do not involve measurement attributes.

Next, we rank the relevant publishers for q according to the channels they can contribute to q . Let P be a relevant publisher and R a relevant republisher, where P has the view $\sigma_{D^\kappa \wedge D^\mu}(r)$ and R has the view $\sigma_{E^\kappa \wedge E^\mu}(r)$. We say that P is *subsumed by* R w.r.t. q , and write $P \preceq_q R$, if

$$D^\kappa \wedge C^\kappa \models E^\kappa.$$

We say that P is *strictly subsumed* by R w.r.t. q , and write $P \prec_q R$, if $P \preceq_q R$, but not $R \preceq_q P$. Note that a producer can never subsume another publisher. The reason is that producer views provide only a sound description of a stream, not a complete one.

We present a method for constructing query plans that consist of publishers that are maximal with regard to the subsumption relation “ \preceq_q ”. We suppose that the publisher configuration and the query $q = \sigma_C(r)$ are fixed.

A relevant publisher is *maximal* if it is not strictly subsumed by another relevant publisher. Let M_q be the set of maximal relevant publishers for q . We partition M_q into the subsets M_q^S and M_q^R , consisting of producers and republishers, respectively.

We write $P_1 \sim_q P_2$ if $P_1 \preceq_q P_2$ and $P_2 \preceq_q P_1$. Clearly, if P_1 and P_2 are two distinct maximal relevant publishers, and $P_1 \preceq_q P_2$, then $P_1 \sim_q P_2$. Note that a producer is never equivalent to another publisher because it cannot subsume the other publisher. Thus, the relation “ \sim_q ” is an equivalence relation on the set of republishers M_q^R and we say that R_1 is *equivalent to* R_2 w.r.t. q if $R_1 \sim_q R_2$.

We denote the equivalence class of a republisher R w.r.t. q as $[R]_q$. Any two equivalent republishers will contribute the same answer tuples satisfying q . Therefore we need only choose one element of any class $[R]_q$ when constructing a plan for q . The set of all equivalence classes of maximal relevant republishers is denoted as

$$\mathcal{M}_q^R = \{ [R]_q \mid R \in M_q^R \}.$$

We call the pair $\mathcal{M}_q = (\mathcal{M}_q^R, M_q^S)$ the *meta query plan* for q . We show next how one can construct actual query plans from \mathcal{M}_q .

A sequence $\langle R_1, \dots, R_k \rangle$ of republishers that is obtained by choosing one representative from each class of republishers in \mathcal{M}_q^R is called a *supplier sequence* for q . Let $\langle R_1, \dots, R_k \rangle$ be a supplier sequence for q and S_1, \dots, S_l be the stream producers in M_q^S . Suppose the descriptive views of the R_i have the conditions D_i . We define the *canonical republisher query* for the sequence as

$$Q^R = \sigma_{C_1}(R_1) \uplus \dots \uplus \sigma_{C_k}(R_k),$$

where $C_1 = C$ and $C_i = C \wedge \neg(D_1 \vee \dots \vee D_{i-1})$ for $i \in 2..k$. Moreover, we define the *canonical stream producer query* as

$$Q^S = \sigma_{C'}(S_1) \uplus \dots \uplus \sigma_{C'}(S_l),$$

where $C' = C \wedge \neg(D_1 \vee \dots \vee D_k)$.

The selection conditions on the disjuncts in Q^R ensure that R_i only contributes channels that no $R_{i'}$ with $i' < i$ can deliver, and the condition C' in Q^S guarantees that producers only contribute channels that cannot be delivered by the republishers.

Note that the conditions C_i depend on the order of republishers in the sequence, but once the order is fixed, they do not depend on which republisher is chosen from an equivalence class. Moreover, although syntactically the conditions C' in Q^S may differ for different supplier sequences, they are all equivalent.

Let us again consider q_2 being posed over \mathcal{P}_1 . The set of relevant publishers is $\{S_2, S_3, S_5, R_1, R_2, R_3\}$, and the set of maximal relevant publishers is $\{S_5, R_1, R_2, R_3\}$. For q_2 , we have that $R_1 \sim_{q_2} R_3$. Thus, we have $\mathcal{M}_{q_2} = (\{\{R_1, R_3\}, \{R_2\}\}, \{S_5\})$. In Fig. 1, the equivalence class $\{R_1, R_3\}$ is shown by the dotted lines. A cost model may be applied to select between the republishers in an equivalence class, e.g. based on response time. From \mathcal{M}_{q_2} we derive

$$\begin{aligned} Q = & \sigma_{\text{tool}=\text{'ping'}} \wedge \text{latency} \leq 10.0(R_1) \uplus \sigma_{\text{tool}=\text{'ping'}} \wedge \text{latency} \leq 10.0 \wedge \neg(\text{from}=\text{'hw'}) (R_2) \\ & \uplus \sigma_{\text{tool}=\text{'ping'}} \wedge \text{latency} \leq 10.0 \wedge \neg(\text{from}=\text{'hw'} \vee \text{from}=\text{'ral'}) (S_5), \end{aligned}$$

as a valid query plan. The meta query plan for q_1 is $\mathcal{M}_{q_1} = (\{\{R_1\}\}, \emptyset)$.

The following theorem, taken from [3], identifies the local queries Q that are plans for a global query q .

Theorem 1. *Let q be a global query, Q^R be the canonical republisher query, and Q^S be the canonical stream producer query for some supplier sequence for q . Then, a plan for q is*

$$Q = Q^R \uplus Q^S.$$

3 Plan Maintenance

When a new publisher is introduced or an existing publisher drops off, the query plans of consumers have to be adapted to the new situation. A meta query plan \mathcal{M}_q depends on two parameters: a global query q , which is explicit in our notation, and a publisher configuration \mathcal{P} , which so far was taken to be fixed. However, during the execution period of a global query, which is usually long lived, it is possible that \mathcal{P} changes to a new configuration \mathcal{P}' because new publishers arise or existing ones vanish. As a consequence, the meta query plan for q in the new configuration \mathcal{P}' may differ from the one in \mathcal{P} and the query plan may have to change as well. To make the dependency on the publisher configuration explicit, in this section we will write meta query plans for q w.r.t. \mathcal{P} and \mathcal{P}' as $\mathcal{M}_q(\mathcal{P})$ and $\mathcal{M}_q(\mathcal{P}')$, respectively.

One possibility to move from $\mathcal{M}_q(\mathcal{P})$ to $\mathcal{M}_q(\mathcal{P}')$ would be to compute the new meta query plan from scratch. However, as we outlined in [8], it is likely to be more efficient to (i) identify when at all $\mathcal{M}_q(\mathcal{P})$ is affected by a change of \mathcal{P} , and (ii) to amend $\mathcal{M}_q(\mathcal{P})$, whenever this is possible, based on the information contained in $\mathcal{M}_q(\mathcal{P})$ and the publisher involved in the change. In the following

we shall investigate formally how *adding* a publisher to \mathcal{P} or *deleting* one affects meta query plans. Without loss of generality, we assume that all publishers added to or deleted from \mathcal{P} are relevant for q , since other changes do not have an effect on the meta query plan.

3.1 Adding a Producer

If a relevant producer S_0 is added then we consider two cases. If S_0 is subsumed w.r.t. q by an existing maximal republisher, say R , then all the data coming from S_0 will be republished by R and, similarly, by every republisher in $[R]_q$, the equivalence class of R . Since the current meta query plan contains the class $[R]_q$, no change is needed. However, if S_0 is not subsumed by a maximal republisher, then it has to be added to the set of maximal relevant producers.

Proposition 1. *Suppose $\mathcal{P}' = \mathcal{P} \cup \{ S_0 \}$. Then:*

1. *if there is a class $[R]_q \in \mathcal{M}_q^{\mathbf{R}}$ such that $S_0 \preceq_q R$, then $\mathcal{M}_q(\mathcal{P}') = \mathcal{M}_q(\mathcal{P})$;*
2. *if there is no such class, then $\mathcal{M}_q(\mathcal{P}') = (\mathcal{M}_q^{\mathbf{R}}, M_q^{\mathbf{S}} \cup \{ S_0 \})$.*

3.2 Deleting a Producer

If a relevant producer S_0 is dropped, then the situation is similar to the previous one. If S_0 is not a maximal relevant producer, that is, if it is subsumed by some republisher, then the meta query plan is not affected by the change, otherwise it has to be removed from the set of maximal relevant producers.

Proposition 2. *Suppose $\mathcal{P}' = \mathcal{P} \setminus \{ S_0 \}$. Then:*

1. *if $S_0 \notin M_q^{\mathbf{S}}$, then $\mathcal{M}_q(\mathcal{P}') = \mathcal{M}_q(\mathcal{P})$;*
2. *if $S_0 \in M_q^{\mathbf{S}}$, then $\mathcal{M}_q(\mathcal{P}') = (\mathcal{M}_q^{\mathbf{R}}, M_q^{\mathbf{S}} \setminus \{ S_0 \})$.*

3.3 Adding a Republisher

The situation becomes more complex when a relevant republisher R_0 is added. There are three possible cases to be considered:

1. R_0 is strictly subsumed by some maximal republisher R , that is, $R_0 \prec_q R$;
2. R_0 is equivalent to some maximal republisher R , that is, $R_0 \sim_q R$; or
3. R_0 is not subsumed by any existing maximal republisher.

In case 1, R_0 is not needed in the meta query plan, while in case 2, R_0 needs to be added to the class $[R]_q$. In case 3, R_0 will form a new equivalence class of its own. Moreover, it may be the case that R_0 subsumes some existing maximal producers and republishers. If it does, then the subsumption is strict and the publishers concerned have to be removed from the meta query plan.

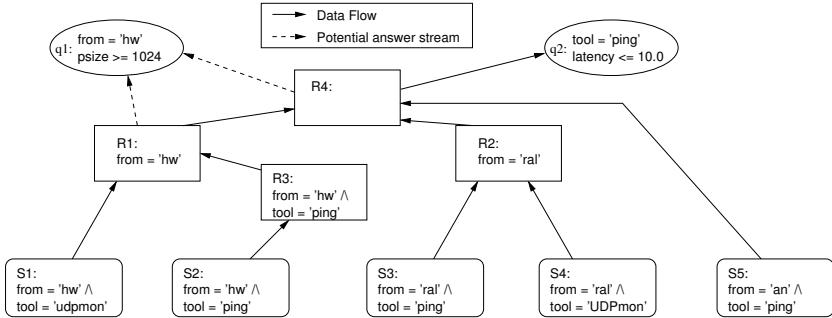


Fig. 2. Consumer query q_1 being posed at the publisher configuration \mathcal{P}_2

Proposition 3. Suppose $\mathcal{P}' = \mathcal{P} \cup \{ R_0 \}$. Then:

1. if there is a class $[R]_q \in \mathcal{M}_q^{\mathbf{R}}$ such that $R_0 \prec_q R$, then $\mathcal{M}_q(\mathcal{P}') = \mathcal{M}_q(\mathcal{P})$;
2. if there is a class $[R]_q \in \mathcal{M}_q^{\mathbf{R}}$ such that $R_0 \sim_q R$, then $\mathcal{M}_q(\mathcal{P}')$ is obtained from $\mathcal{M}_q = (\mathcal{M}_q^{\mathbf{R}}, M_q^{\mathbf{S}})$ by replacing the class $[R]_q$ in $\mathcal{M}_q^{\mathbf{R}}$ with $[R]_q \cup \{R_0\}$;
3. if there is no class $[R]_q \in \mathcal{M}_q^{\mathbf{R}}$ with $R_0 \preceq_q R$, then $\mathcal{M}_q(\mathcal{P}') = (\mathcal{M}_q^{\mathbf{R}'}, M_q^{\mathbf{S}'})$ where
 - $\mathcal{M}_q^{\mathbf{R}'}$ is obtained from $\mathcal{M}_q^{\mathbf{R}}$ by adding the class $\{R_0\}$ and removing all classes $[R']_q$ with $R' \preceq_q R_0$
 - $M_q^{\mathbf{S}'}$ is obtained from $M_q^{\mathbf{S}}$ by only keeping the producers that are not subsumed by R_0 , i.e.

$$M_q^{\mathbf{S}'} = \{ S \in M_q^{\mathbf{S}} \mid S \not\preceq_q R_0 \}.$$

Let us illustrate this by adding a republisher $R_4 := \sigma_{true}(\text{ntp})$, which gathers all tuples published for the ntp relation, to the publisher configuration \mathcal{P}_1 . With regard to consumer query q_1 , we are in case 2 since $R_4 \sim_{q_1} R_1$. So R_4 would be added to the equivalence class of R_1 in \mathcal{M}_{q_1} . There is no change to the query plan of q_1 as the old plan is consistent with the new meta query plan. However, for q_2 we are in case 3 as R_4 subsumes w.r.t. q_2 all the maximal relevant publishers. The situation in the new configuration \mathcal{P}_2 is illustrated in Fig. 2.

3.4 Deleting a Republisher

Similar to the previous situation, we distinguish three cases when a republisher R_0 is dropped:

1. R_0 is strictly subsumed by some maximal relevant republisher;
2. R_0 is equivalent to some other maximal relevant republishers; or
3. R_0 is not subsumed by any existing maximal republisher.

In case 1, the meta query plan is not affected, while in case 2, the republisher R_0 needs to be deleted from its equivalence class. Case 3, by contrast, requires more

action. The reason is that, intuitively, the deletion of R_0 leaves a hole in the set of data that can be delivered by the remaining republishers in the meta query plan.

To “patch” the hole, those relevant republishers need to be identified that were not maximal in the presence of R_0 , but are promoted to maximal ones after the demise of R_0 . We define the *patch* of $M_q^{\mathbf{R}}$ for R_0 as the set M' consisting of those republishers R' relevant for q where (i) $R' \prec_q R_0$ and (ii) there is no $R \in M_q^{\mathbf{R}} \setminus \{R_0\}$ such that $R' \prec_q R$. Then the new set $M_q^{\mathbf{R}'}$ is obtained by removing the class $[R_0]_q$ from $M_q^{\mathbf{R}}$ and adding the classes obtained from the elements of M' . Moreover, some producers that were subsumed by R_0 may not be subsumed by the newly promoted maximal republishers and have to be added to the set $M_q^{\mathbf{S}}$ to yield $M_q^{\mathbf{S}'}$.

Proposition 4. Suppose $\mathcal{P}' = \mathcal{P} \setminus \{R_0\}$. Then:

1. if $R_0 \notin M_q^{\mathbf{R}}$, then $\mathcal{M}_q(\mathcal{P}') = \mathcal{M}_q(\mathcal{P})$;
2. if $R_0 \in M_q^{\mathbf{R}}$ and there is another $R \in M_q^{\mathbf{R}}$ with $R_0 \sim_q R$, then $\mathcal{M}_q(\mathcal{P}')$ is obtained from $\mathcal{M}_q = (\mathcal{M}_q^{\mathbf{R}}, M_q^{\mathbf{S}})$ by replacing the class $[R]_q$ in $\mathcal{M}_q^{\mathbf{R}}$ with $[R]_q \setminus \{R_0\}$;
3. if $R_0 \in M_q^{\mathbf{R}}$ and the class $[R_0]_q \in \mathcal{M}_q^{\mathbf{R}}$ is a singleton, then $\mathcal{M}_q(\mathcal{P}') = (\mathcal{M}_q^{\mathbf{R}'}, M_q^{\mathbf{S}'})$ where
 - $\mathcal{M}_q^{\mathbf{R}'}$ is obtained from $\mathcal{M}_q^{\mathbf{R}}$ by removing the class $\{R_0\}$ and adding all classes $[R']_q$ such that $R' \in M'$ is in the patch M' of $M_q^{\mathbf{R}}$ for R_0
 - $M_q^{\mathbf{S}'}$ is obtained from $M_q^{\mathbf{S}}$ by adding those producers relevant for q that were subsumed by R_0 , but are not subsumed by any republisher in $\mathcal{M}_q^{\mathbf{R}'}$.

We now consider removing republisher R_1 from the publisher configuration \mathcal{P}_2 to create \mathcal{P}_3 . Republisher R_1 is a maximal relevant publisher for q_1 which is equivalent w.r.t. q_1 to R_4 . This means that we are in case 2 above. Hence, we replace the equivalence class $\{R_1, R_4\}$ in $\mathcal{M}_{q_1}(\mathcal{P}_2)$ with the equivalence class $\{R_4\}$ to give the meta query plan $\mathcal{M}_{q_1}(\mathcal{P}_3) = (\{\{R_4\}\}, \emptyset)$.

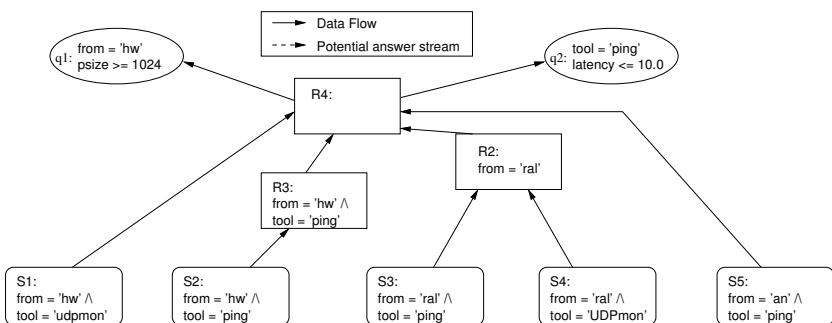


Fig. 3. Consumer query q_1 being posed at the publisher configuration \mathcal{P}_3

The situation in configuration \mathcal{P}_3 is shown in Fig. 3. We note that in \mathcal{P}_3 the consumer query q_1 no longer has a choice in the publisher to contact to retrieve its answer stream in the most efficient manner. It must contact republisher R_3 , hence the line showing the data flow from R_3 to q_1 is now solid in Fig. 3.

We also note that since the consumer had been using R_1 in its query plan it will need to “switch” to a new query plan using R_3 . Mechanisms on how to switch between query plans are discussed in Section 5.

3.5 Discussion

The propositions above show that plan maintenance is straightforward if *producers* come or go and is more complicated when the set of relevant *republishers* changes. The reason is that the streams of producers are only sound w.r.t. their descriptive views, but republisher streams are both sound and complete. As a consequence, a republisher can replace other publishers, which is impossible for a producer. In Section 5 we discuss the implications of maintaining query plans during the execution of a query.

The cost of performing the plan maintenance operations is polynomial in the number of publishers involved, providing that subsumption can be checked in polynomial time. From the cases encountered with the R-GMA system this is often the case since conditions can only contain conjunctions. Of course, if conditions can contain disjunctions then the problem is NP-hard.

4 Maintaining a Hierarchy of Republishers

Similar to consumers, republishers should include other republishers in their query plans. This leads to a *hierarchy* of republishers through which data streams can flow. A straightforward approach would be to construct and maintain plans and meta plans for republishers in the same way as for consumers. However, a simple example shows that this does not work.

Consider the publisher configuration \mathcal{P}_0 consisting of the publishers S_1 , R_1 , and R_4 , as defined above. Applying the planning and maintenance techniques developed for consumers would result in the republishers having the meta query plans $\mathcal{M}_{R_1} = (\{\{R_4\}\}, \emptyset)$, and $\mathcal{M}_{R_4} = (\{\{R_1\}\}, \emptyset)$. The only corresponding query plans are $Q_{R_1} = \sigma_{\text{from}=\text{'hw'}}(R_4)$, and $Q_{R_4} = \sigma_{\text{true}}(R_1)$.

The resulting hierarchy is unsatisfying for two reasons: (i) the publishers are *not connected* to the producer, and (ii) there is a *cycle in the dependency relation* of the republishers in that R_1 consumes from R_4 and vice versa. Obviously, the first fact prevents the republishers from obtaining any data. Moreover, if there are cyclic dependencies between republishers, tuples could travel an infinite number of times along the cycle. The volume of the resulting stream could grow indefinitely, and the stream would be neither duplicate free nor weakly ordered.

4.1 Requirements for a Publisher Hierarchy

To come up with a mechanism for constructing and maintaining query plans for republishers, we identify properties that such a mechanism should have.

We expect that, similar to the one for consumer queries, query plans for republishers are unions of selections over publishers and that, in addition, each republisher has a meta query plan, from which actual plans can be formed. Moreover, we expect that the mechanism produces some kind of meta query plan that contains a set of candidate publishers on which actual plans are based.

The query plans executed by the republishers define a dependency relation among the publishers which we call the *physical hierarchy*, that is, one through which the data flows. The meta query plans of the republishers define a more general dependency relation which we call the *logical hierarchy*.

We argue that the following four requirements are essential for any planning and maintenance mechanism for republishers:

Correctness: The plan for each republisher should be sound and complete for the defining query as well as duplicate free and weakly ordered.

Cycle Freeness: Neither the physical nor the logical hierarchy should contain any cycles.

Uniqueness of the Logical Hierarchy: The logical hierarchy should only depend on the publisher configuration \mathcal{P} . The way in which it has been created, that is, the order in which publishers have been added and deleted should have no influence on it.

Local Query Planning: To create its query plan and meta query plan, a republisher should not need any information about the plans and meta plans of other republishers.

Clearly, a plan that is not correct would fail to implement the republisher and can lead to the republishers being disconnected from the producers, as in the example above. The physical hierarchy of the example contains a cycle. We discussed the negative effects of that cycle. Since cycles in the logical hierarchy may give rise to cycles in the physical hierarchy, they need to be ruled out, too. A logical hierarchy will be much easier to understand if it depends only on the structure of a configuration and not on its history. For the physical hierarchy, which is a subrelation of the logical hierarchy, a republisher should be allowed to form it as it sees fit. If query planning is local, republishers only need to communicate with the registry service and not with other republishers.

4.2 Generating and Maintaining Query Plans for Republishers

A general analysis shows that cycles and missing links to producers as in the example above are a consequence of the definition of relevant publishers in Subsection 2.3. To avoid cycles, for two republishers R_1 and R_2 it should be impossible that R_1 is relevant for R_2 and at the same time R_2 is relevant for R_1 .

To refine the concept of relevance, we introduce a new, general, subsumption relation. Let P be a publisher with the view $\sigma_{D^\kappa \wedge D^\mu}(r)$ and R be a republisher with the view $\sigma_{E^\kappa \wedge E^\mu}(r)$. We say that P is *subsumed* by R , and write $P \preceq R$, if

$$D^\kappa \models E^\kappa \quad \text{and} \quad E^\mu \models D^\mu.$$

Intuitively, this means that R delivers tuples for a channel if P does so, and that for its channels, P delivers all the values that R delivers. We say that P is *strictly subsumed* by R , and write $P \prec R$, if $P \preceq R$, but not $R \preceq P$. Clearly, if both R_1 is subsumed by R_2 and R_2 by R_1 , then the view conditions of R_1 and R_2 are logically equivalent.

We now use this notion of general subsumption to modify the definition of relevance. We say that a producer is *strongly relevant* if it is relevant for the query of R_0 and that a republisher R is *strongly relevant* for R_0 if $R \prec R_0$. One readily verifies that strong relevance implies relevance.

Let us reconsider the example from the beginning of the section. It is immediate to see that in \mathcal{P}_0 , both S_1 and R_1 are strongly relevant for R_4 while only S_1 is strongly relevant for R_1 . If instead of relevant publishers, we only admit strongly relevant ones to query planning, the meta query plan for R_1 in \mathcal{P}_0 is $\mathcal{M}_{R_1}(\mathcal{P}_0) = (\emptyset, \{S_1\})$, while the one for R_3 is $\mathcal{M}_{R_4}(\mathcal{P}_0) = (\{\{R_1\}\}, \emptyset)$. The corresponding query plans are $Q_{R_1} = \sigma_{\text{from}=\text{'hw'}}(S_1)$ and $Q_{R_4} = \sigma_{\text{true}}(R_1)$. Neither the physical nor the logical hierarchy contain a cycle. The next proposition shows that this is not accidental.

Proposition 5. *If meta query plans for republishers are only based on strongly relevant publishers, then all plans derived from them are correct. Moreover, for any publisher configuration, there is a unique logical hierarchy, which is cycle free, and meta query plans and plans can be computed locally by each republisher.*

Intuitively, the result holds for the following reasons. Plans are still correct, as they were for consumer queries, because the relevance criterion for producers has not been changed. A plan using only strongly relevant republishers may have to access more producers than one relying on relevant republishers, because fewer producers are made redundant by republishers. By definition, the logical hierarchy depends only on the publishers and their conditions, which uniquely determine it. Since any strongly relevant publisher for R is strictly subsumed by R , there cannot be any cycles in the logical hierarchy. A plan can be computed by an individual republisher based on information about its strongly relevant publishers, without coordinating the planning with other republishers.

The meta query plan for a consumer or republisher query q is defined in terms of relevant or strongly relevant publishers, respectively, that are maximal w.r.t. the quasiorder “ \preceq_q ”. A closer inspection of the results in Section 3 reveals that all four propositions hold, independently of how relevant publishers are defined. Therefore, the maintenance techniques of that section can be applied directly for republisher queries.

5 Implementing the Query Plans

The theory developed for generating and maintaining query plans does not prescribe how it should be implemented, i.e. which components are responsible for the various stages of the planning process. We motivated the work with the development of R-GMA [3,4], a Grid information and monitoring system, which partially implements a stream integration system. However, the current R-GMA system only allows a republisher to collect together data streams and store them. They cannot be used to publish a data stream and so cannot be used to form a publisher hierarchy.

The final stage of developing the query planning mechanisms is to decide the responsibilities of the components and develop communication protocols between the components to enable them to (i) plan for a global query, (ii) detect when the publisher configuration is altered and maintain the plans, and (iii) switch between query plans. We do this in the context of an experimental implementation based on the approach followed by the R-GMA system. We anticipate that the results will be used to further enhance the capabilities of R-GMA.

5.1 Computing Query Plans

The registry uses a database to store details of every publisher and consumer registered in the system. These details include a structured representation of the conditions of the queries, so they need only be parsed once. This is possible as R-GMA has a limited continuous query language: continuous queries are selections with conditions that are conjunctions of the form attr op value , where op may be $<$, \leq , $=$, \geq , $>$.

When a consumer is constructed, it contacts the registry with details of its global query. The registry encodes the satisfiability and entailment tests needed to find the set of relevant publishers as a single SQL query over its database. The registry then returns to the consumer a list of all relevant publishers. From this list of relevant publishers, the consumer first constructs a meta query plan and then derives a query plan, as described in Section 2.3.

5.2 Maintaining Query Plans

The task of maintaining query plans can be broken down into two stages. The first stage is to detect when the publisher configuration has changed. Once a change has been detected, the maintenance mechanisms of Section 3 can be applied.

When a publisher is created, it will contact the registry to make itself known to the system. Thus, the registry will know when a new publisher is created. However, due to the distributed nature of the system, there must be some mechanism that can detect when a publisher is no longer contactable, e.g. due to a network failure.

A simple timeout mechanism based on tuples being delivered from a publisher to a consumer is not adequate. For example, it is possible that a consumer should

consume from a republisher but the republisher does not have any sources. To avoid unnecessary plan maintenance, we require that every component periodically sends the registry a heartbeat message. Failure of this message to arrive means that the component is no longer available for queries.

Once the registry has detected that there is a change in the publisher configuration, it can construct an SQL query over its database to find all the consumers and republishers which are affected, and inform them of the change. The consumers and republishers first check whether the publisher is maximal relevant for its query and if it is they apply the results of Section 3 to test whether they need to (i) amend their meta query plans, and (ii) if they have altered their meta query plan, to check whether their current query plan is consistent with the new meta query plan.

Since consuming components are (i) initially informed of all relevant publishers, and (ii) informed of any changes in the set of relevant publishers, they can maintain their meta query plans in all the cases presented in Section 3 without further interaction with the registry. To patch their plan as required by Proposition 4 case 3 the component need only refer to the list of relevant publishers.

5.3 Switching Between Query Plans

In developing protocols for switching between query plans we have focused on the properties of the query plan. That is, the protocols must ensure, as far as possible in a distributed setting, that answer streams are sound and complete w.r.t. their defining query, weakly ordered, and duplicate free.

There are five scenarios when a query plan is not consistent with the new meta query plan.

1. A producer is added to the meta query plan.
2. A producer is removed from the meta query plan.
3. A republisher is removed from the meta query plan which (i) appears in an equivalence class with other republishers, and (ii) is in the query plan.
4. A republisher is added to the meta query plan which has created a new equivalence class.
5. A republisher is removed from the meta query plan which was in an equivalence class on its own.

The situations in cases 1 and 2 have already been implemented in R-GMA. For case 1, each publisher caches a published tuple for a duration defined by the publisher's *retention period*. This provides the registry with the time needed to contact the consumers and republishers for which the producer is relevant and for them to then contact the producer and start streaming. For case 2, the producer is simply removed from the query plan.

The situation in case 3 is the one presented in the example of Section 3.4 when republisher R_1 is removed from publisher configuration \mathcal{P}_2 . Since q_1 had been using R_1 in its query plan, then it must switch to using republisher R_4 .

We require that a consumer maintains a *latest-state* buffer where it stores the most recent tuple received on each channel. When switching from R_1 to R_4 ,

the consumer searches in its latest-state buffer for the oldest tuple t_o received from R_1 . The consumer then requests that R_4 starts streaming from t_o^τ , the timestamp of t_o . We must use the timestamp of t_o rather than the tuple itself as our streams are only weakly ordered.

Upon receiving this message, R_4 consults the tuples in its publishing buffer and, providing that t_o^τ is still within its retention period, starts streaming all tuples with a timestamp equal to, or newer than t_o^τ . Otherwise it will start streaming from the oldest tuple in its buffer. On receiving the stream from R_4 , the consumer must filter, on a per channel basis, the first part of the stream against its latest-state buffer. Only once it starts receiving tuples newer than the ones in its latest-state buffer does its answer stream start getting new tuples.

This mechanism ensures that the answers received by the consumers are sound w.r.t. the query. Providing that the tuples are still within the retention periods of the publishers involved, the answer stream will be complete. In the cases where the stream is not complete, a bound in time can be provided on the incompleteness in the answer stream, i.e. the time between t_o^τ and the re-publisher's retention period. Due to the filtering based on the latest-state buffer the answer stream will be duplicate free, and weak order is guaranteed by construction. We decided to search for the oldest tuple since the retention period is defined by the publisher, and in R-GMA it is desirable to set these to large periods.

The same mechanism can be applied in cases 4 and 5 except on a per publisher basis. Some additional care needs to be taken with the changing of conditions to existing publishers in the query plan.

6 Related Work

Publish/subscribe systems provide middleware for managing the communication of events asynchronously [6]. A subscription is a simple filter on the events published to the system. The publish/subscribe system must process every event as it arrives and forward it on to the relevant subscriptions. Several systems have been developed, including Rebecca [7] and SIENA [2]. Our approach differs as (i) events are streamed from the publisher directly to the consumer, and (ii) we allow more expressive subscriptions.

Work on processing and querying data streams has resulted in centralised data stream management systems, e.g. STREAM [1], and Telegraph [10]. The StreamGlobe system [12] looks at processing data streams in a P2P environment. However, it is unclear as to the guarantees of the correctness of the answer streams. Another approach to the distributed filtering of data streams is dQUOB [11]. However, each data source must then publish for a different relation.

7 Conclusions

In this paper, we have built upon earlier techniques for integrating data streams in a publish/subscribe network [3,8] which allowed consumers to generate a query

plan and identified that such plans need to be updated when the set of publishers changes. We have now formalised the techniques for maintaining these query plans when a publisher is added to or removed from the network.

We have also discussed, and presented an approach to, the problem of switching between query plans whilst ensuring that the answer stream remains sound and complete w.r.t. the query, duplicate free, and weakly ordered.

Other interesting areas of research would be to increase the complexity of allowable queries, e.g. selection queries that join tables together using sliding windows, and to allow more expressive view conditions, e.g. containing joins.

References

1. A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. *Data-Stream Management: Processing High-Speed Data Streams*, chapter STREAM: The Stanford Data Stream Management System. Springer-Verlag, 2005. To appear.
2. A. Carzaniga and A.L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM*, pages 163–174, Karlsruhe (Germany), August 2003.
3. A. Cooke, A.J.G. Gray, and W. Nutt. Stream integration techniques for grid monitoring. *Journal on Data Semantics*, 2:136–175, 2005.
4. A.W. Cooke, A.J.G. Gray, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Cordeonsi, R. Byrom, L. Cornwall, A. Djaoui, L. Field, S.M. Fisher, S. Hicks, J. Leake, R. Middleton, A. Wilson, X. Zhu, N. Podhorszki, B. Coghlan, S. Kenny, D. O’Callaghan, and J. Ryan. The relational grid monitoring architecture: Mediating information about the grid. *Journal of Grid Computing*, 2(4):323–339, December 2004.
5. K. Czaikowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *10th International Symposium on High Performance Distributed Computing*, pages 181–194, IEEE Computer Society, June 2001. IEEE Computer Society.
6. P.T. Eugster, P.A. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.
7. L. Fiege, G. Mühl, and F.C. Gärtner. A modular approach to build structured event-based systems. In *Proc. 2002 ACM Symposium on Applied Computing*, pages 385–392, Madrid (Spain), March 2002. ACM Press.
8. A.J.G. Gray and W. Nutt. Republishers in a publish/subscribe architecture for data streams. In *Proc. 22nd British National Conference on Databases*, volume 3567 of *LNCS*, pages 179–184, Sunderland (UK), July 2005. Springer-Verlag.
9. A. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.
10. S.R. Madden and M.J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proc. 18th International Conference on Data Engineering*, pages 555–566, San Jose (CA, USA), February 2002. IEEE Computer Society.
11. B. Plale and K. Schwan. Dynamic querying of streaming data with the dQUOB system. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):422–432, April 2003.

12. B. Stegmaier, R. Kuntschke, and A. Kemper. StreamGlobe: Adaptive query processing and optimization in streaming P2P environments. In *Proc. of the 1st International Workshop on Data Management for Sensor Networks*, pages 88–97, Toronto (Canada), August 2004. VLDB.
13. B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A Grid monitoring architecture. Global Grid Forum Performance Working Group, March 2000. Revised January 2002.
14. G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.

Containment of Conjunctive Queries with Arithmetic Expressions

Ali Kiani and Nematollaah Shiri

Dept. of Computer Science & Software Engineering,
Concordia University, Montreal, Quebec, Canada
`{ali_kian, shiri}@cse.concordia.ca`

Abstract. We study the problem of query containment for conjunctive queries with arithmetic constraints (QWAE). Such queries arise naturally in conventional database applications, information integration, and cooperative information systems. Given two such queries Q_1 and Q_2 , we propose an algorithm that decides the containment $Q_2 \sqsubseteq Q_1$. The proposed algorithm returns a QWAE Q'_2 obtained by rewriting Q'_2 such that $Q'_2 \sqsubseteq Q_2$. This provides partial answers to the QWAE Q_1 , which would otherwise be discarded by existing standard or extended techniques for query containment.

1 Introduction

Query containment (QC) is a fundamental problem in a large number of database applications including answering queries using views [9, 1, 7], query rewriting [12, 13, 7], data integration [15], and query optimization [5].

A query Q_2 is said to be contained in a query Q_1 (denoted by $Q_2 \sqsubseteq Q_1$), if there is a containment mapping (CM) μ from Q_1 to Q_2 . When $Q_2 \sqsubseteq Q_1$, it means that the answer to Q_2 on any database D is contained in the answer to Q_1 on D , i.e., $Q_2(D) \subseteq Q_1(D)$.

Query containment has been studied extensively in database research investigating different aspects and applications [4, 8, 6, 2]. Most of these work consider conjunctive queries (CQ), also known as Select-Project-Join (*SPJ*) queries; Cartesian product is a special case of join, where there is no common attributes. More recently, there have been some interesting proposals for containment of conjunctive queries extended with constraints, in the form of built-in predicates [2, 8]. To set the stage and motivate this work, in what follows we consider three examples of common SQL queries and their corresponding conjunctive queries.

Example 1. Let $r(A, B)$ and $s(C, D, E)$ be relation schemas. Consider the following SQL queries:

q_1 : `SELECT A, B
FROM r;`

q_2 : `SELECT A, B
FROM r, s
WHERE r.B = s.C;`

These SQL queries q_1 and q_2 can be expressed as the following conjunctive queries Q_1 and Q_2 , respectively.

$$\begin{aligned} Q_1(A, B) &::= r(A, B). \\ Q_2(A, B) &::= r(A, B), s(B, D, E). \end{aligned}$$

Taking as the containment mapping μ , the identity function which maps each variable to itself, we can see that $Q_2 \sqsubseteq Q_1$.

Klug [8] and Afrati et. al. [2], extended the problem of query containment to support conjunctive queries with constraints. The constraints they consider compare a single variable with a variable or a constant. Example 2 includes examples of such queries in SQL.

Example 2. For relations r and s defined in Example 1, consider the following SQL queries.

$$\begin{aligned} q_1: \text{SELECT } A, B \\ \text{FROM } r \\ \text{WHERE } B > 10; \\ q_2: \text{SELECT } A, B \\ \text{FROM } r, s \\ \text{WHERE } r.B = s.C \text{ AND } B > 15; \end{aligned}$$

These SQL queries can be expressed as the following conjunctive queries with attribute comparisons.

$$\begin{aligned} Q_1(A, B) &::= r(A, B), B > 10. \\ Q_2(A, B) &::= r(A, B), s(B, D, E), B > 15. \end{aligned}$$

These are examples of the conjunctive queries with attribute comparisons, studied in [8, 2]. According to their results, we can find a containment mapping from Q_1 to Q_2 , based on which the predicate part of the body of Q_1 becomes a subset of the body of Q_2 and constraint in Q_2 implies constraint in Q_1 , i.e., $(B > 15) \Rightarrow (B > 10)$. In this case, therefore, we can conclude that $Q_2 \sqsubseteq Q_1$.

Example 3. Let us now consider again the same relations but the SQL queries:

$$\begin{aligned} q_1: \text{SELECT } A \\ \text{FROM } r, s \\ \text{WHERE } r.B = s.C \text{ AND} \\ r.B = s.D + s.E; \\ q_2: \text{SELECT } A \\ \text{FROM } r, s \\ \text{WHERE } r.B = s.C \text{ AND} \\ r.B = s.D \text{ AND } s.E = 0; \end{aligned}$$

These queries can be expressed as the following conjunctive queries with arithmetic expressions.

$$\begin{aligned} Q_1(A) &::= r(A, B), s(B, D, E), B = D + E. \\ Q_2(A) &::= r(A, B), s(B, D, 0), B = D. \end{aligned}$$

It is not hard to convince ourselves that $Q_2 \sqsubseteq Q_1$. However, to the best of our knowledge such cases were not studied in earlier research. Example 3 motivates our work in this paper in which we extend the problem of query containment to support conjunctive queries with arithmetic expression, or QWAE for short.

Below is another example of QWAEs which shows the containment for conjunctive queries with constraints in the form of arithmetic expressions.

Example 4. Consider relations $r_1(X, M)$ and $r_2(X, N)$, and conjunctive queries with arithmetic expressions below.

$$\begin{aligned} Q_1(X, M) &::= r_1(X, M), \quad r_2(X, N), \quad M + N > 0. \\ Q_2(X, 10) &::= r_1(X, 10), \quad r_2(X, 0). \end{aligned}$$

Using the CM μ from Q_1 to Q_2 which maps X to itself and M to 10, we can see that it is in fact the case that $Q_2 \sqsubseteq Q_1$. In this paper, we generalize earlier work on query containment with attribute comparisons and present an algorithm for deciding containment in the presence of arithmetic expressions. Moreover, given such queries Q_1 and Q_2 , if $Q_2 \not\sqsubseteq Q_1$, our proposed algorithm finds a rewriting Q'_2 , based on Q_2 such that $Q'_2 \sqsubseteq Q_1$.

The organization of the rest of this paper is as follows. Next, we review related work. In section 3, we discuss the concept of containment of conjunctive queries and query containment in the presence of constraints in the form of arithmetic expressions. In section 4, we present a decision algorithm to test containment of such queries. The proposed algorithm also finds a contained rewriting. Concluding remarks and future work are discussed in section 5.

2 Related Work

Chandra and Merlin studied query containment (QC) for standard conjunctive queries [4]. They showed that for two conjunctive queries Q_1 and Q_2 , $Q_2 \sqsubseteq Q_1$ if and only if there is a containment mapping from Q_1 to Q_2 . Such a mapping, maps a constant to itself, and a variable to either a variable or a constant. Under this mapping, the head of the queries will become identical, and each subgoal of Q_1 becomes a subgoal in Q_2 [2]. Millstein et al. [11] consider query containment in the context of information integration and introduced the concept of relative containment for cases where sources become unavailable. Gupta et al. [6], and Benedikt et al. [3] studied constraint queries and the problem of constraint satisfaction without performing a complete constraint evaluation. Klug [8] and Afrati et al. [2] consider the problem of QC in the presence of arithmetic comparisons and propose an algorithm for finding and testing containment. Furthermore, [2] studied cases in which query normalization is not required, which is the first step in containment checking algorithm. Hence, their algorithm reduces the size of the query and provides efficiency. They also take advantage of an inequality graph to find the homomorphism for the subject queries. The following categorization from [6] classifies different languages using which we can represent constraints in the queries.

1. Conjunctive queries (CQs)([4]).
2. Unions of conjunctive queries ([14]).
3. Conjunctive queries with arithmetic comparisons ([8, 2]).
4. Conjunctive queries with negated subgoals ([10]).
5. Other combinations of the above, that is, CQs or unions of CQs, with or without arithmetic comparisons and with or without negated subgoals.
6. Recursive datalog, including cases with or without arithmetic comparisons and with or without negated subgoals.

Although [8] and [2] both considered the containment problem for queries with constraints, their work focused more on comparing single attributes, that is constraints of the form $A\theta E_r$, in which A is an attribute, θ is a comparison operator, and E_r is either a an attribute or a constant. In this work, we consider conjunctive queries with constraints of the form $E_l\theta E_r$, in which E_l and E_r are *arithmetic expressions* and $\theta \in \{<, >, =\}$. Next, we define some concepts and notations.

3 Query Containment and Conjunctive Queries with Arithmetic Expressions

In this section, we study the problem of containment of conjunctive queries with arithmetic expressions.

Queries with arithmetic constraints can be divided into the four categories, as follows. For each category we also provide an example.

1. Constraints with equality of atomic operands:

$$Q(X, Y) :- r_1(X, Y, Z), r_2(X, M), Y = 1999, M = Z.$$

2. Constraints with inequality of atomic operands:

$$Q(X, Y) :- r_1(X, Y, Z), r_2(X, M), Y > 1999, M \leq Z.$$

3. Constraints with equality on expressions:

$$Q(X, Y) :- r_1(X, Y, Z), r_2(X, M), Z = 2 \times M, X + M = Y + Z.$$

4. Constraints with inequality on expressions:

$$Q(X, Y) :- r_1(X, Y, Z), r_2(X, M), Y + Z < M.$$

Note that such queries are common in SQL. Also note that while cases 1 and 2 deal with comparison of single attributes, which in a sense are special case of arithmetic expressions, cases 3 and 4 involve more general case of arithmetic expressions.

3.1 Conjunctive Queries with Arithmetic Expressions

Now, we formally define the conjunctive queries with arithmetic expression (QWAE), and their normal form.

A conjunctive query with arithmetic expression is a statement of the form:

$$h(\bar{X}) : - g_1(\bar{X}_1), \dots, g_k(\bar{X}_k), \alpha_1, \dots, \alpha_n,$$

where $g_i(\bar{X}_i)$ are subgoals and α_j are constraints. Let A be the set of attributes and constants in the query. Each predicate g_i in the rule body is a *base relation*, and every argument in g_i is an expression over A . Every constraint α_j is of the form $E_l \theta E_r$, where E_l (left) and E_r (right) are arithmetic expressions over A and θ is a comparison operator. Examples of such queries are as follows.

1. $p_1(X, T) :- r_1(X, H), r_2(X, N, R), T = H \times R + N \times 100.$

Note that in this query, the head attribute T is defined as an expression over attributes in the body. This is equivalent to the SQL query below, indicating how such queries may arise in applications.

```
SELECT r1.X, H × R + N × 100 AS T
FROM r1, r2
WHERE r1.X = r2.X;
```

2. $p_2(X) :- s_1(X, V_1, C_1, Y), s_1(X, V_2, C_2, Y_1), V_1 \times C_1 = V_2 - C_2, Y_1 = Y + 1.$

In this query, the constraint part filters out the tuples for which the constraint $V_1 \times C_1 = V_2 - C_2$ is not satisfied. The SQL version of this query would be as follows:

```
SELECT s1.X
FROM s1, s2
WHERE s1.X = s2.X AND s1.V1 × s1.C1 = s2.V2 - s2.C2;
```

To sum up, we can see that arithmetic expressions play two roles in QWAEs. One is to define an attribute in the head and the other is to express a condition/constraint in the rule body. Later, we will show that deciding containment of Q_2 in Q_1 could be done easier if the constraints in Q_2 do not include “disjunction”, that is, the comparison operators θ s used in Q_2 are restricted to $\{\leq, \geq, \neq\}$. Note that when $\theta \in \{\leq, \geq, \neq\}$, it implies the presence of disjunction, e.g., $a \leq b$ can be represented as $(a < b) \vee (a = b)$, or $a \neq b$ can be represented as $(a < b) \vee (a > b)$.

3.2 Normalized Queries

Normalization of the queries is the first step in the algorithms proposed for deciding query containment [4, 2]. This is because a containment mapping is a function that associates with each variable, a variable or a constant. However, after normalizing the two queries, their ordinary subgoals will have distinct variables, making it possible to map variables in Q_1 to variables in Q_2 . Therefore, normalization is required in general, however, it might result in higher cost as the number of containment mappings increases in general for normalized queries.

Definition 1. *[Normalized QWAE]*

A QWAE is said to be normalized if every argument in every normal predicate in the head or body of the query is a unique variable not repeated in any other normal predicates.

The following algorithm, adapted from [2], can be used to normalize QWAEs.

1. For all occurrences of an expression exp (which is not a single attribute) in the head or subgoals, replace the occurrence of exp by a new distinct variable X_i , and add $X_i = exp$ to the body as a new constraint β .
2. For every argument X (atomic expression) in the subgoals in the rule body, replace all but the first occurrence of X by a new distinct variable X_i , and add $X_i = X$ to the body as a new constraint β .

3. For every argument X (atomic expression) in the head, replace all but the first occurrence of X by a new distinct variable X_i , and add $X_i = X$ to the body as a new β .

To illustrate how this algorithm works, consider the following conjunctive query, provided earlier as an example in category 3. It should be clear that normalization generates an equivalent query.

$$Q_1(X, X) \vdash r_1(X, Y, Z), r_2(X, Y + Z), X = Y - Z$$

Step 1:

$$\begin{aligned} Q_1(X, X) \vdash & r_1(X, Y, Z), r_2(X, X_1), X = Y - Z, \\ & X_1 = Y + Z. \end{aligned}$$

Step 2:

$$\begin{aligned} Q_1(X, X) \vdash & r_1(X, Y, Z), r_2(X_2, X_1), X = Y - Z, \\ & X_1 = Y + Z, X_2 = X. \end{aligned}$$

Step 3:

$$\begin{aligned} Q_1(X, X_3) \vdash & r_1(X, Y, Z), r_2(X_2, X_1), X = Y - Z, \\ & X_1 = Y + Z, X_2 = X, X_3 = X. \end{aligned}$$

3.3 Containment of Conjunctive Queries

Next, we recall the notion of containment of conjunctive queries.

Definition 2. [Containment of Conjunctive Queries] A conjunctive query Q_2 is contained in a conjunctive query Q_1 , denoted $Q_2 \sqsubseteq Q_1$, if and only if there is a containment mapping μ from Q_1 to Q_2 , such that μ maps the attributes of Q_1 to the attributes of Q_2 , and after applying μ , the heads are unified and the predicates in the body of Q_1 form a subset of the predicates in the body of Q_2 .

3.4 Containment in the Presence of Constraints

The following result from [8, 6, 2], characterizes the containment of conjunctive queries with constraints. Let Q_1 and Q_2 be conjunctive queries defined as follows:

$$\begin{aligned} Q_1(\bar{X}) \vdash & g_1(\bar{X}_1), \dots, g_k(\bar{X}_k), \alpha_1, \dots, \alpha_n, \\ Q_2(\bar{Y}) \vdash & g_1(\bar{Y}_1), \dots, g_l(\bar{Y}_l), \beta_1, \dots, \beta_m. \end{aligned}$$

Let $C = \{\alpha_i | i \in [1..n]\}$ and $D = \{\beta_j | j \in [1..m]\}$ be the set of constraints in Q_1 and Q_2 , respectively. Then, $Q_2 \sqsubseteq Q_1$ iff $D \Rightarrow \mu_1(C) \vee \dots \vee \mu_q(C)$, where μ_p s are containment mappings from Q_1 to Q_2 making the head unified and the predicates in the body of Q_1 a subset of the predicates in the body of Q_2 .

Example 5. [Contained query] Consider the following pair of QWAE queries:¹

$$Q_1(X, Y) \vdash r_1(X, Y), r_2(X, Z), X = 20 - 2 \times Z.$$

¹ We will revisit these queries in Example 9 to illustrate the steps of our algorithm.

$$Q_2(M, N) \dashv r_1(M, N), r_2(M, P), M + N = 20, N = 2 \times P.$$

Let $\mu = \{X \rightarrow M, Y \rightarrow N, Z \rightarrow P\}$ be a containment mapping. It is straightforward to see that $(M + N = 20, N = 2 \times P) \Rightarrow (M = 20 - 2 \times P)$, and hence $Q_2 \sqsubseteq Q_1$.

As illustrated in [2], it is possible that $Q_2 \sqsubseteq Q_1$, while every component $\mu_i(C)$ in the implication $D \Rightarrow \mu_1(C) \vee \dots \vee \mu_q(C)$ is false. To see this, consider the following implications:

$$\begin{aligned} I_1 &: (X < a \vee X > a) \Rightarrow X < a. \\ I_2 &: (X < a \vee X > a) \Rightarrow X > a. \\ I_3 &: (X < a \vee X > a) \Rightarrow (X < a \vee X > a). \end{aligned}$$

Here, implications I_1 and I_2 are not always true, whereas I_3 is. Regarding the constraint in queries, this happens when D “includes” disjunction. Assume that $D = D_1 \vee \dots \vee D_k$. Then we have the following equivalences in which $C' = \mu_1(C) \vee \dots \vee \mu_q(C)$.

$$\begin{aligned} D \Rightarrow \mu_1(C) \vee \dots \vee \mu_q(C) &\equiv \\ D_1 \vee \dots \vee D_k \Rightarrow \mu_1(C) \vee \dots \vee \mu_q(C) &\equiv \\ D_1 \vee \dots \vee D_k \Rightarrow C' &\equiv \\ \neg(D_1 \vee \dots \vee D_k) \vee C' &\equiv \\ (\neg D_1 \wedge \dots \wedge \neg D_k) \vee C' &\equiv \\ (\neg D_1 \vee C') \wedge \dots \wedge (\neg D_k \vee C') &\equiv \\ (D_1 \Rightarrow C') \wedge \dots \wedge (D_k \Rightarrow C') &\equiv \\ (D_1 \Rightarrow \mu_1(C) \vee \dots \vee \mu_q(C)) \wedge \dots \wedge (D_k \Rightarrow \mu_1(C) \vee \dots \vee \mu_q(C)). \end{aligned}$$

Let us refer to implication $D_i \Rightarrow \mu_1(C) \vee \dots \vee \mu_q(C)$ as η_i . Note that here the original implication would be true if and only if η_i is true, for all i . Assume that for every η_i , there exists a $\mu_j(C)$ such that $D_i \Rightarrow \mu_j(C)$. Note that even though we cannot conclude the implication $(D_1 \vee D_2 \vee \dots) \Rightarrow \mu_j(C)$, the following implication holds: $(D_1 \vee \dots \vee D_k) \Rightarrow (\mu_1(C) \vee \dots \vee \mu_q(C))$. This shows while there is no single $\mu_i(C)$ in the original implication for which the implication $D \Rightarrow \mu_i(C)$ is satisfied, the implication as a whole might be satisfied.

Definition 3. [Containment of conjunctive queries with arithmetic expression] Let $Q_1(\bar{X})$ and $Q_2(\bar{Y})$ be two normalized QWAES defined as follows.

$$\begin{aligned} Q_1(\bar{X}) \dashv g_1(\bar{X}_1), \dots, g_k(\bar{X}_k), \alpha_1, \dots, \alpha_n. \\ Q_2(\bar{Y}) \dashv g_1(\bar{Y}_1), \dots, g_l(\bar{Y}_l), \beta_1, \dots, \beta_m. \end{aligned}$$

Let $C = \{\alpha_i | i \in [1..n]\}$ and $D = \{\beta_j | j \in [1..m]\}$. Then, Q_2 is contained in Q_1 denoted as $Q_2 \sqsubseteq Q_1$ iff $D \Rightarrow \mu_1(C) \vee \dots \vee \mu_q(C)$, where μ_i s are containment mappings from Q_1 to Q_2 making the predicates in the body of Q_1 a subset of the predicates in the body of Q_2 , and the heads identical.

Proposition 1. Let Q_1 and Q_2 be QWAES, and suppose $C = \{\alpha_i | i \in [1..n]\}$ and $D = \{\beta_j | j \in [1..m]\}$ are constraints in the bodies, respectively. Also suppose μ_p s

are containment mappings from Q_1 to Q_2 , and D includes only conjunctions. That is, each β_j is of the form $E_l \theta E_r$, where $\theta \in \{<, >, =\}$. Then, $Q_2 \sqsubseteq Q_1$ if and only if $D \Rightarrow \mu_i(C)$ for some $i \in [1..q]$.

Proof. Suppose D consists of conjunctions only, i.e., $D = D_1 \wedge \dots \wedge D_k$. Thus,

$$\begin{aligned} D \Rightarrow \mu_1(C) \vee \dots \vee \mu_q(C) &\equiv \\ D_1 \wedge \dots \wedge D_k \Rightarrow \mu_1(C) \vee \dots \vee \mu_q(C) \end{aligned}$$

Suppose for some i and j , $D_i \Rightarrow \mu_j(C)$. Then we have, $D_1 \wedge \dots \wedge D_k \Rightarrow \mu_j(C)$, where $i \in [1..k]$, and hence $D_1 \wedge \dots \wedge D_k \Rightarrow \mu_1(C) \vee \dots \vee \mu_q(C)$. This means that from $D_i \Rightarrow \mu_j(C)$, we may conclude that $D \Rightarrow \mu_j(C)$.

In definition 3, if assume that D does not include disjunction, then according to proposition 1, condition $D \Rightarrow \mu_1(C) \vee \dots \vee \mu_q(C)$ reduces to $D \Rightarrow \mu_i(C)$, for some $i \in [1..q]$. Therefore, we only need to test $D \Rightarrow \mu(C)$, for a containment mapping μ found. This explains why in this work, comparison operators considered are restricted to $>$, $<$, and $=$. This is further discussed when we present our algorithm TC, for testing containment.

3.5 Categories of Containment w.r.t Arithmetic Expressions

Let Q_1 and Q_2 be any pair of QWAES, and C and D be their sets of constraints, respectively, where D includes only conjunction. Suppose T is the set of tuples (in any given database) satisfying predicates in the body of Q_2 . If there exists a containment mapping μ from Q_1 to Q_2 , there are three possible cases we distinguish for containment of Q_2 in Q_1 , described as follows.

1. $T_D \subseteq T_{\mu(C)}$.

That is, tuples T_D satisfying constraint D also satisfy constraint $\mu(C)$, denoted as $T_{\mu(C)}$. In other words, D implies $\mu(C)$, and hence $Q_2 \sqsubseteq Q_1$. Example 5 falls into this category.

2. $T_D \cap T_{\mu(C)} = \emptyset$.

In this case, we should simply discard Q_2 , since it is not contained in Q_1 nor it can be used to find a query that is contained in Q_1 .

3. $T_D \cap T_{\mu(C)} \neq \emptyset$.

That is, some tuples satisfying constraint D also satisfy constraint $\mu(C)$. In this case, $Q_2 \not\sqsubseteq Q_1$. However, since there are still some common tuples in T_D and $T_{\mu(C)}$, we can define a query V' such that $V' \sqsubseteq Q_1$. For this, we look for the set of constraints that are not implied by D . We call this set as *not implied constraints* (*NIC*, for short). Assume $C_1 \subseteq \{\mu(\alpha_1), \dots, \mu(\alpha_n)\}$. Let $C_2 = \{\gamma_i | \gamma_i \in C_1, D \not\Rightarrow \gamma_i\}$. That is, C_2 contains all the constraints in C_1 that are not implied by D . We refer to C_2 as NIC_{Q_2} of Q_1 .

Let C_3 be the subset of C_2 that contains the constraints γ which are not defined over \bar{X} . That is, C_3 contains those constraints in γ in which there is at least one attribute that does not appear in \bar{X} . We then try to reformulate the constraints in C_3 , using a constraint solver, such that the attributes in C_3 are among those

mentioned in \bar{X} . Each constraint obtained in this reformulation is added into C_4 together with the other NIC constraints. If C_3 is empty or we could reformulate all its elements, then V' could be defined as:

$$V'(\bar{Y}) \text{-} V(\bar{Y}), \gamma_1, \dots, \gamma_u.$$

in which $\gamma_i \in C_4$. Otherwise, there is no containment to consider, and hence $V' \sqsubseteq Q_1$.

Example 6. [Partially contained query] Consider the QWAES defined as follows:

$$\begin{aligned} Q(X, Y, Z) &\text{-} r_1(X, Y), r_2(X, Z), X + Y = Z. \\ V(M, N, P) &\text{-} r_1(M, N), r_2(M, P), 2 \times N = 3 \times P. \end{aligned}$$

Let $\mu = \{X \rightarrow M, Y \rightarrow N, Z \rightarrow P\}$. It is easy to verify that $D \not\Rightarrow \mu(C)$, where $D = \{2 \times N = 3 \times P\}$ and $\mu(C) = \{M + N = P\}$. Therefore, $V \not\sqsubseteq Q$. However, in this case, we test if we can restrict V to get a contained query. If this is the case, we can produce some partial answer by defining V' such that $V' \sqsubseteq Q$. The test result is positive in this case and we define V' as follows, using which we can conclude that $V' \sqsubseteq Q$.

$$V'(M, N, P) \text{-} V(M, N, P), M + N = P.$$

3.6 Unifiable and Non-unifiable Head Predicates

In standard query containment, for Q_2 to be contained in Q_1 , it is required that the head of queries Q_1 and Q_2 be unifiable. However, in this work, even when the heads are not unifiable, we still check if Q_2 can be used in finding a contained query for Q_1 . For this, we define the concept of head transformation as follows.

Definition 4. [Head Transformation]

Given two conjunctive queries $Q_1(\bar{X})$ and $Q_2(\bar{Y})$, if there exists a function λ from \bar{Y} to \bar{X} such that $\forall X_i$ in \bar{X} , $X_i = \lambda(\bar{Y})$, then λ is a transformation from the head of Q_2 to the head of Q_1 . Intuitively, when λ exists, it means that \bar{X} is computable on the basis of \bar{Y} .

Note that such λ is directional, and the transformation is not based on the identity of attribute names but based on a containment mapping between the attributes in the the two queries.

Example 7. [Unifiable Heads] Consider the following queries:

$$\begin{aligned} Q(A) &\text{-} r(X, Y), A = X + Y. \\ V(M, N) &\text{-} r(M, N). \end{aligned}$$

Furthermore, let us consider the containment mapping $\mu = \{X \rightarrow M, Y \rightarrow N\}$ from Q to V . Here, A can be expressed as an expression over M, N . Accordingly, there is a transformation from the arguments of $V(M, N)$ to that of $Q(A)$, attribute A in this case. That is, $A = M + N$. In this case, even though V is not contained in Q , we can define the following query V' which is contained in Q .

$$V'(A) \text{-} V(M, N), A = M + N.$$

Note that now the structure of the answer tuples through V' matches that of query Q .

Example 8. [Non-unifiable Heads] Let us next consider the following QWAES.

$$\begin{aligned} Q(X, Y) &:- r(X, Y). \\ V(A) &:- r(X, Y), A = X + Y. \end{aligned}$$

Note that even though the body of V is contained in the body of Q , V is not contained in Q , in any sense, standard or extended. The reason is that, because of the particular structure of the heads, there is no transformation from variables in $V(A)$ to those in $Q(X, Y)$. In fact, X and Y come from r which we do not have access to, considering V as a view/source.

Proposition 2. *Let Q_1 and Q_2 be two normalized QWAE queries such that there is at least one attribute in the head of Q_1 for which we cannot find a one to many mapping to attributes of Q_2 . Then Q_2 is not contained in Q_1 nor can be used to find a contained query (which includes only Q_2 as a predicate).*

Note that 1-1 mapping is a special case of one to many mapping. Also having a 1-1 and onto mapping from the attributes of Q_1 to those in Q_2 corresponds to the standard case of containment. So, according to proposition 2, we may discard containment mappings based on which there is no transformation from Q_2 to Q_1 .

4 Testing Containment

In this section, we introduce an algorithm for testing containment of QWAE queries. We will refer to this algorithm as *testing containment*, or **TC**, for short.

Algorithm $\text{TC}(Q_1, Q_2, R)$:

Input: Q_1 and Q_2 conjunctive queries with arithmetic expressions.

Output: R , a rewriting of query Q_2 which is contained in Q_1 .

Method:

S_1 . Normalize Q_1 and Q_2 . Now the two queries have the following form:

$$\begin{aligned} Q_1(\bar{X}) &:- g_1(\bar{X}_1), \dots, g_k(\bar{X}_k), \alpha_1, \dots, \alpha_n. \\ Q_2(\bar{Y}) &:- g_1(\bar{Y}_1), \dots, g_l(\bar{Y}_l), \beta_1, \dots, \beta_m. \end{aligned}$$

Let $\bar{X} = \{A_1, \dots, A_i\}$ and $\bar{Y} = \{B_1, \dots, B_j\}$.

S_2 . Find a new containment mapping μ from Q_1 to Q_2 . In this step, we do not consider the heads when finding μ . If no containment mapping exists, return \emptyset and *halt*.

S_3 . Check condition below for *Proposition 2*.

For all $A_i \in \bar{X}$, there exists exp_i such that $A'_i = \mu(A_i) = \text{exp}_i$, where exp_i is an expression defined over \bar{Y} . If this condition does not hold, go to step S_2 .

- S_4 . For constraints $C = \{\alpha_1, \dots, \alpha_n\}$ in Q_1 and $D = \{\beta_1, \dots, \beta_m\}$ in Q_2 , determine the category of containment as described in section 3.5.
- If it falls in category 2, then $D \not\Rightarrow C$, and hence $Q_2 \not\subseteq Q_1$. Go to S_2 .
 - If it falls in category 1, then $Q_2 \sqsubseteq Q_1$. Return the following query:

$$R(\bar{X}) :- Q_2(\bar{Y}), A_1 = exp_1, \dots, A_f = exp_f,$$

where A_i 's are attributes in \bar{X} and exp_i s are as found in step S_3 .

Note that in this case, if $\bar{Y} = \mu(\bar{X})$, then Q_2 is contained in Q_1 . That is by applying the containment mapping, the head of Q_1 unifies the head of Q_2 .

- If it falls in category 3, then $Q_2 \not\subseteq Q_1$, however based on Q_2 , we define a query V' which is contained in Q_1 .

$$V'(\bar{Y}) :- Q_2(\bar{Y}), \gamma_1, \dots, \gamma_u,$$

where γ_i 's are those constraints in Q_1 that can not be implied by constraints in Q_2 . Note that as explained in category 3 of containment, some constraint γ might have attributes that are not in the head of Q_2 or Q_1 . We then try to reformulate them based other constraints as described in category 3 of containment. If this could be done such that no such γ remain, then every constraint is well defined, and we return the following query. Otherwise go to step S_2 .

$$R(\bar{X}) :- V'(\bar{Y}), A_1 = exp_1, \dots, A_f = exp_f,$$

where A_i 's are attributes in \bar{X} and exp_i s are as defined in step S_3 .

End TC.

The following examples introduced earlier illustrate the steps of this algorithm.

Example 9. Assume Q and V are QWAE queries, defined as follows:

$$\begin{aligned} Q(X, Y) &:- r_1(X, Y), r_2(X, Z), X = 20 - 2 \times Z. \\ V(M, N) &:- r_1(M, N), r_2(M, P), M + N = 20, N = 2 \times P. \end{aligned}$$

S_1 . [Normalizing Q and V]

$$\begin{aligned} Q(X, Y) &:- r_1(X, Y), r_2(X_1, Z), X = 20 - 2 \times Z, X_1 = X. \\ V(M, N) &:- r_1(M, N), r_2(M_1, P), M + N = 20, N = 2 \times P, M_1 = M. \end{aligned}$$

- [Finding a containment mapping] Consider a CM $\mu = \{X \rightarrow M, Y \rightarrow N, Z \rightarrow P, X_1 \rightarrow M_1\}$. Using a constraint solver, we can see that, $(M + N = 20, N = 2 \times P, M_1 = M) \Rightarrow (M = 20 - 2 \times N, M_1 = M)$.
- [Checking the heads] Using the containment mapping μ , we can derive that $X = M$ and $Y = N$, so the heads unify.
- [Determining the containment category] Since we could find for both X and Y , mapping expressions over M and N that map the head of Q to the head of V , then Q and V fall in the first category of containment

$(V \sqsubseteq Q)$, and as the result the proposed algorithm returns the following query, in which $V(M, N)$ is the view defined above.

$$R(X, Y) :- V(M, N).$$

Below is an example of a query V which is not contained in Q in the usual sense, however, the algorithm determines a query R , such that $R \sqsubseteq Q$

Example 10. Let Q and V be QWAE queries defined as follows:

$$\begin{aligned} Q(A) &:- r_1(X, Y), r_2(X, Z), 3 \times X = Y, A = X + Y. \\ V(M, N) &:- r_1(M, N), r_2(M, P), 2 \times N = 3 \times P. \end{aligned}$$

S_1 . We first normalize Q and V , which gives:

$$\begin{aligned} Q(A) &:- r_1(X, Y), r_2(X_1, Z), 3 \times X = Y, A = X + Y, X_1 = X. \\ V(M, N) &:- r_1(M, N), r_2(M_1, P), 2 \times N = 3 \times P, M_1 = M. \end{aligned}$$

S_2 . Consider $\mu = \{X \rightarrow M, Y \rightarrow N, Z \rightarrow P, X_1 \rightarrow M_1\}$.

S_3 . Since we can find the following mapping between A, M , and N , we may continue with μ , which yields $A = M + N$.

S_4 . Since $D = \{2 \times N = 3 \times P, M_1 = M\}$ does not imply $C_1 = \{3 \times M = N, M_1 = M, A = M + N\}$, that is, $D \not\Rightarrow (3 \times M = N)$, we should consider category 3 of containment. That is, $V \not\sqsubseteq Q$. In this case, however, we can define V' which is contained in Q . Here, $C_1 = \{3 \times M = N, M_1 = M, A = M + N\}$ and $C_2 = \{3 \times M = N\}$.

Thus, $C_3 = C_4 = \{3 \times M = N\}$. Hence, we obtain V' and R as follows:

$$\begin{aligned} V'(M, N) &:- V(M, N), 3 \times M = N, \\ R(A) &:- V'(M, N), A = M + N. \end{aligned}$$

4.1 Complexity Analysis

The complexity of our TC algorithm above can be analyzed noting the two steps involved, i.e., (1) finding containment mappings and (2) solving the constraints in order to decide their implication.

It is known that finding containment mapping is NP [2], On the other hand, the complexity of the constraint solver depends on the constraint type. As shown in [7], in case of linear equality constraints, the complexity of deciding implication of constraints is $O(n^3)$, where n is the number of variables in the constraints. For non-linear constraints, the complexity would be higher. However, since dealing with constraints is the second step in the algorithm after a containment mapping is being found, the type of constraints cannot be used to restrict the complexity of TC as a whole, and hence it is at least NP.

5 Conclusions and Future Work

Containment of conjunctive queries has been studied extensively and is the core concept in database research including answering queries using views, query rewriting,

ing, data integration, query optimization, etc. Recently, standard query containment has been extended to conjunctive queries with constraints in the form of attribute comparisons. In this work, we considered the containment problem for queries with arithmetic expressions, which we called QWAE, which extends earlier proposals for conjunctive queries with constraints. A main motivation of this work was to support applications such as information integration, answering queries using views, cooperative query answering, to name a few.

We identified three cases of containment of QWAEs and discussed details of query containment in the presence of constraints and also cases in which arguments of the predicates, in the head and/or the body, are not necessarily single attributes. We also extended the concept of normalized queries and proposed an algorithm for normalizing queries with arithmetic expressions, adapted from [2].

Finally we proposed an algorithm TC that given two QWAE queries Q_1 and Q_2 , it returns a query, that is a rewriting of Q_2 which is contained in Q_1 .

An implementation of this work is underway. Moreover, based on this work, we have developed an algorithm for answering queries using views in the context of conjunctive queries with linear equality constraints which is a special case of QWAEs [7]. We are currently investigating ways to extend our algorithm deveoped in [7] for answering queries using views to support more general QWAEs. As another future work, we are working on a generic model for information integration, which would be a platform for the result of the aforementioned works. In a more broad view, the algorithm TC together with our view-based query answering algorithm would best suit in our generic model for information integration to support query processing.

Acknowledgements

This work was in part supported by grants from Natural Sciences and Engineering Research Council (NSERC) of Canada, and from ENCS, Concordia University. We also thank anonymous reviewers for their useful comments.

References

1. Afrati, Foto; Li, Chen; and Mitra Prasenjit. Answering queries using views with arithmetic comparisons. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 209–220. ACM Press, 2002.
2. Afrati, Foto; Li, Chen; and Mitra Prasenjit. On containment of conjunctive queries with arithmetic comparisons. In *EDBT*, pages 459–476, 2004.
3. Benedikt Michael and Libkin Leonid . Safe constraint queries. In *In Proc. ACM Symp. on Principles of Database Systems*, pages 99–108, 1998.
4. Chandra A.K. and Merlin P.M. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9th Annual ACM Symp. on the Theory of Computing*, pages 77–90, 1977.
5. Chaudhuri, Surajit; Krishnamurthy, Ravi; Potamianos, Spyros; and Shim, Kyuseok. Optimizing queries with materialized views. In *Proc. IEEE Int. Conf. on Data Eng.*, pages 190–200, 1995.

6. Gupta, Ashish; Sagiv, Yehoshua; Ullman, Jeffrey D.; and Widom, Jennifer. Constraint checking with partial information. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 45–55, Minneapolis, Minnesota, 1994.
7. Kiani, Ali and Shiri, Nematollaah. Answering queries in heterogenous information systems. In *Proc. of ACM Workshop on Interoperability of Heterogeneous Information Systems*, Bremen, Germany, Nov. 4, 2005.
8. Klug A. On conjunctive queries containing inequalities. *Journal of the ACM*, pages 35(1): 146–160, 1988.
9. Levy, Alon Y. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.
10. Levy, Alon Y. and Sagiv, Yehoshua. Queries independent of updates. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, pages 171–181, Dublin, Ireland, 1993.
11. Millstein, Todd; Levy, Alon; and Friedman, Marc. Query containment for data integration systems. *J. Comput. Syst. Sci.*, 66(1):20–39, 2003.
12. Pottinger, Rachel and Levy, Alon Y. A scalable algorithm for answering queries using views. In *The VLDB Journal*, pages 484–495, 2000.
13. Qian, Xiaolei. Query folding. In *Proc. IEEE Int'l Conf. on Data Eng.*, New Orleans, LA, February 1996.
14. Sagiv, Y. and Yannakakis, M. Equivalence among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1981.
15. Ullman, Jeffrey D. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.

Multiagent Negotiation for Fair and Unbiased Resource Allocation

Karthik Iyer and Michael Huhns

Department of Computer Science and Engineering, University of South Carolina,
Columbia SC 29208, USA
`{iyerk, huhns}@engr.sc.edu`

Abstract. This paper proposes a novel solution for the n agent cake cutting (resource allocation) problem. We propose a negotiation protocol for dividing a resource among n agents and then provide an algorithm for allotting portions of the resource. We prove that this protocol can enable distribution of the resource among n agents in a fair manner. The protocol enables agents to choose portions based on their internal utility function, which they do not have to reveal. In addition to being fair, the protocol has desirable features such as being unbiased and verifiable while allocating resources. In the case where the resource is two-dimensional (a circular cake) and uniform, it is shown that each agent can get close to $1/n$ of the whole resource.

1 Introduction

Autonomous agents are currently being given responsibilities in numerous applications from the economic, industrial, commercial, social, and entertainment sectors of the world. Autonomy means that the agents have a high degree of freedom and choice in initiating actions on their own, planning goals for themselves, and taking actions to achieve them. In order to achieve their goals agents must have sufficient resources and capabilities. But what if the resources are insufficient for all agents to achieve their goals? This typically is the case where an agent lives in a multiagent system and has to share resources with other agents, coordinate its own tasks to resolve conflicts, and sometimes persuade other agents to do things for it. All this could be done centrally by one designer, who could chalk out every detail of the entire system. The central agency would then be able to resolve conflicts, divide resources, plan schedules, etc. But this is not applicable in open systems where individual agents are designed by different designers and there is no concept of a central agency. Multiagent systems are inherently distributed, heterogeneous, and open. So what could be done in situations like this?

In the real world, people from different backgrounds come together and resolve conflicts or form alliances for mutual benefits. This sort of interaction between people is based on negotiation. Negotiation [1] is “when two parties strike a deal through argumentation or arbitration for the mutual good of both.”

Negotiation as per Davis and Smith [2] is “A discussion in which interested parties exchange information and come to an agreement.” This work revolves around a resource allocation problem and the main concern of agents participating in the negotiation scheme described is to ensure a fair and unbiased allocation of a resource.

In open multiagent systems there is generally no global control, no globally consistent knowledge, and no globally shared goals or success criteria. So there exists a real competition among agents, which act to maximize their own utilities. We assume all the utility functions are private to the agents. The negotiation protocol should be immune to information hiding and lying by agents. This has to be ensured as there is no control on the design of agents that interact in open environments and the only check that could be made is by cleverly designing their interaction mechanism. We describe our negotiation protocol that tries to incorporate the desirable characteristics mentioned above. We use the problem domain of cake cutting in order to better visualize and formalize our solution for an n -agent resource allocation protocol.

2 Background

The cake-cutting problem is a well known example of resource sharing among rational agents. The classical solution for the two-person case, divide and choose, was first proposed by Steinhaus [3]. This solution, where one person divides the cake into two pieces and the other gets first choice of a piece, is both fair and envy free. However, it has been difficult to scale up the solution to n agents. One of the solutions [4] for dividing the cake among n agents fairly has been to use a moving knife parallel to one of the edges of the cake. The knife cuts when one of the agents yells “Cut!” and the portion traversed by the knife so far is allotted to the agent. This is an elegant and clean n agent solution for creating n portions fairly in $n-1$ cuts. But the moving knife solution has its own set of drawbacks. First, it is not envy-free. Second, it requires the presence of an unbiased mediator who holds the knife and moves it along the cake at a constant rate. Despite this safeguard, it is difficult to verify the cutting of the cake. For example, if one of the agents alleges that the mediator cut the cake an inch shorter than he had expected, it would be difficult to find out who is telling the truth. In a distributed system, synchronization problems may also occur. An agent with a slower connection to the mediating authority will find its bid to cut may reach later than an agent with a faster connection and may consequently lose the bid. Third, the moving knife protocol is also not pareto optimal. A scheme [5] to improve efficiency in the pareto optimal sense has been proposed with the use of two moving knives. However, the solution works only for division of the resource between two agents, and the agent that moves the knives must be able to estimate the utility function of the other agent well.

Other solutions to n -person division attempt to create a protocol that does not require assistance from an outsider. One way is to scale up from the two-person solution and iteratively add new agents until all have allocations. For $n=2$, the classic divide and choose is used. When agent 3 is added, agents 1 and 2 each divide their portions into 3 parts. Agent 3 then picks one part from each of the other agents. This continues as each agent is added. The drawback for this protocol is that the earlier agents will be faced with the chore of repeatedly dividing their portion into many pieces. The agent that is added last will get its share by doing the least amount of work.

Another solution [6] converts the n agent division problem into many $n-1$ agent problems and then recurses. The recursive calls return when the many two-agent problems are resolved and the answers back up to the top-level call. The drawback is that an agent whose shares remain unallocated till the end has to continuously re-bid for scattered pieces until the iterations end.

A divide-and-conquer procedure [7] instructs the agents to cut the cake into half according to their measure. Then the cuts are ordered and the first $n/2$ cuts are allotted the left half of the cake. The rest are allotted the right half. This procedure continues until 2 agents have to cut the cake where the well known divide-and-choose algorithm can be implemented. An obvious drawback of this procedure is that the number of agents needs to be a power of two, which is an unrealistic requirement.

In the next section we present an improved protocol for n -agent resource division. First we discuss the case of dividing a linear resource, such as a rectangular piece of cake or property along a coastline. Then we show that we can use this protocol for allocating a two-dimensional resource, such as a circular cake, time slots in a 24 hour cycle, or any resource that does not have a well defined starting point. Finally we describe an improved version of the protocol specifically for a circular resource, which removes some of the drawbacks of the linear case.

3 Dividing a Linear Resource Among n Agents

3.1 Protocol

The problem of dividing a linear resource among n agents can be solved by following the protocol below. An example of how a strip of property along a coastline is distributed among three agents is discussed in [8] and [9]. The protocol for the agents is simple: They are required to make $n-1$ marks of the property that delimit n intervals. To the agent making these marks, each of these intervals should be worth $1/n$ of the whole piece and is equally desirable. The procedure guarantees that given such a set of marks, each agent will be guaranteed one of the intervals it has marked. The protocol is fair because each agent will be given one of the intervals it marked.

Refer to figure 1 to see the functioning of the protocol. Given n agents who need to divide a resource among themselves, they approach a mediator (which could be one of the bidding agent themselves) to get their portions of the resources allocated. They register with the mediator, informing it of their interest in participating. After the mediator confirms their participation, the agents submit their bids as a set of marks denoting equal-value portions in their estimation. The mediator waits for all the agents to submit their bids. Then it calculates portions to be allocated to each agent and finally informs each agent of the portion allocated to it.

Theorem 1. If there are n agents and each agent makes $n-1$ marks, creating n portions of a linear cake, then our protocol guarantees that each agent will be allotted a piece of the cake, such that the piece was one of the n portions created by the agent itself.

Proof: This proof assumes that the left and right portions of the cake have been allotted to the agents whose marks came first on the left side and right side respectively. We concentrate on allotting pieces of the cake to the agents still remaining after this procedure. Note that after the first 2 agents have received their share, the marks of the remaining agents need not start at the same point. Now we will have $n-2$ agents with each agent having at least $n-1$ marks creating at least $n-2$ pieces of the cake. Without losing generality, we can add 2 to the above numbers and re-state the problem as:

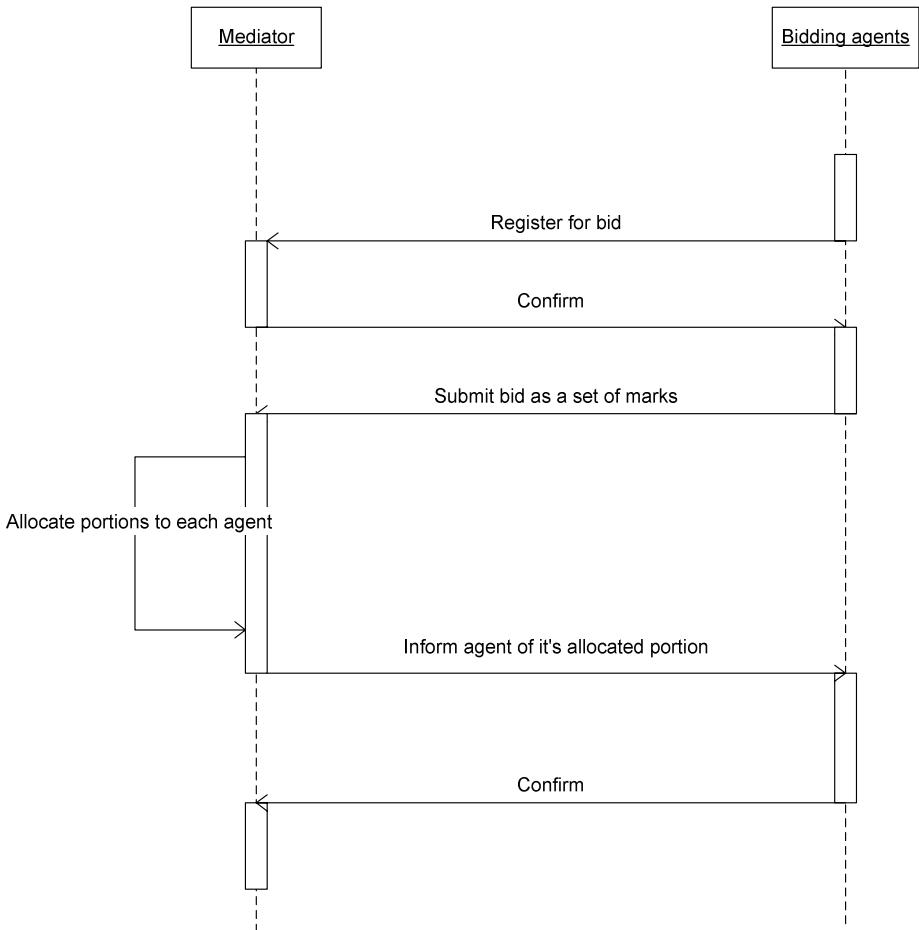


Fig. 1. Linear cake cutting protocol

There are n agents with each agent having at least $n+1$ marks creating at least n pieces or intervals of the cake. It is proved that each agent will be guaranteed a piece of the cake such that the piece was marked by the agent himself.

When agents create marks (that represent their cuts), the following possibilities exist for a particular chosen interval.

1. Pure interval, i.e., no other agent's interval intersects this interval
2. Mixed interval
 - 2.1. The current interval intersects another interval. It does not completely contain any other interval.
 - 2.2. The current interval contains at least one interval made by at least one agent. In addition, there will definitely be an interval that partially intersects the current interval.

Base Case ($n=2$). Each agent will create 2 intervals, each using 3 marks. Find the first mark. Say this mark belongs to agent A. The next mark may or may not be made by A.

1. If the next mark is made by A, this is a pure interval, so allocate the current interval to A. Remove all marks to the left of this interval. Remove all marks made by agent A. We will be left with marks made by agent B to the right of the current interval. Repeat the above procedure. None of B's marks have been deleted yet. Hence the procedure is guaranteed to find an interval to allocate to B, as no other agents are left.
2. If the next mark is made by B, A's interval is mixed. Either case 2.1 or 2.2 will occur.
 - 2.1. A's interval intersects partially with B. In this case allocate the interval to A. Remove all marks to the left of interval. Remove all of A's marks. Since A's interval intersected with B's interval, the previous step will reduce B's marks to 2. Since B has two marks left demarcating an interval and no other agent is remaining, the procedure is guaranteed to find an interval for B.
 - 2.2. A contains at least one interval of B. Since there are no more agents, such an interval of B is guaranteed to be a pure interval and is allocated to B. Remove all marks to the left of the interval. Remove all of B's marks. Since A contained B, the previous step will reduce A's marks to 2. Since A has two marks left demarcating an interval and no other agent is remaining, the procedure is guaranteed to find an interval for A.

This proves that A and B can be allocated fair shares, no matter how the marks are arranged.

For any n ($n>2$). Let us assume that the allocation procedure works for up to k agents. We will show that the procedure works for $k+1$ agents. Each agent will create $k+1$ intervals, each, using $k+2$ marks. Find the first mark. Say this mark belongs to agent i . The next mark might or might not be made by i .

1. If the next mark is made by i , this is a pure interval, allocate current interval to i . Remove all marks to the left of this interval. Remove all marks made by agent i . None of the marks made by other agents will be removed as this was a pure interval for agent i . Hence we will be left with the $k+2$ marks and $k+1$ intervals made by each of the k agents to the right of the current interval. Delete the leftmost mark of each of the k agents. Thus each agent is left with $k+1$ marks and k intervals. This transforms into the allocation procedure for k agents, which we know works. Hence proved.
2. If the next mark is not made by i , suppose the mark belongs to agent j . Add i to the list of agents whose mark has already been seen. Repeat the allocation procedure with j 's mark as the first mark. At some point we will encounter a mark made by one of the agents already in the list. Say the first such agent is l . The interval demarcated by l will not contain any other agent's interval. It may or may not partially intersect with other agent's intervals.
 - 2.1. If l 's interval does not intersect with any other interval, then allocate interval to l . Remove all marks to the left of this interval. Remove all marks made by agent l . The previous step will remove at most one mark of the agents. Other agents will have $k+2$ marks and $k+1$ intervals. Start from the beginning of the

list and as each mark is encountered, check if the mark belongs to an agent already in the agent list. If so, ignore the mark; otherwise add the agent to the list and delete the mark. This step guarantees that we will have k agents, each with k intervals and $k+1$ marks. This transforms into the allocation procedure for k agents, which we know works. Hence proved.

- 2.2. If l 's interval partially intersects with some other interval, then allocate the interval to l . Remove all marks to the left of this interval. Remove all marks made by agent l . The previous step will remove at most one mark of the agents. Other agents will have $k+2$ marks and $k+1$ intervals. Start from the beginning of the list and as each mark is encountered, check if the mark belongs to an agent already in the agent list. If so, ignore the mark; otherwise, add the agent to the list and delete the mark. This step guarantees that we will have k agents each with k intervals and $k+1$ marks. This transforms into the allocation procedure for k agents, which we know works. Hence proved.

This proves that the allocation procedure works for n agents, for any $n \geq 2$.

3.2 Features

If, rather, the resource is circular in shape (e.g., time slots in a 24 hour cycle or a circular cake), can we use the above procedure to get a solution? Henceforth we use the example of a circular cake to explain the division of a circular resource. A quick way to apply the linear protocol to a circular cake is the following.

The protocol begins with a mediator making a mark on the cake denoting it as a starting point. Now take a strip of length equal to the circumference of the cake. Wrap the strip around the cake such that the start and end points of the strip meet at the point marked on the cake. This will show agents the relative locations of their favored pieces. Lay out the strip on a flat surface and let n agents make $n-1$ marks on it demarcating n intervals.

Next, use the allocation procedure for the linear problem to allocate intervals to agents. Take the strip that now shows the intervals allocated to the agents and wrap it around the cake with the start and end of the strip meeting at the point marked on the cake. Make radial portions of the cake according to the intervals marked on paper and allot them to the agents based on their markings. This protocol divides a circular cake among n agents in a fair manner. It has the following features:

- 1) The protocol is fair because each agent gets one of the pieces it demarcated for itself.
- 2) It might not be envy-free, because the initial mark made by the mediator might break up a region of interest for some agents into start and end parts. Such agents will end up at a disadvantage over the others and might envy that the allocation procedure tilted in favor of other agents.
- 3) The linearization makes it difficult for the agents to visualize which portion of the cake they really wanted to have. The portions of the circular cake have to be converted to linear lengths on the strip, which can be non-trivial.

Attempting to apply the linear protocol on a circular cake will end up creating a bias, because it requires a starting point to be declared, which might favor one of the agents. Is there an *unbiased* way to allocate portions of a circular cake? That is the topic of the next section.

4 A Fair and Unbiased Protocol to Partition a Circular Resource Among n Agents

4.1 Protocol

The protocol that needs to be followed by the agents is fairly simple. Given a circular cake, an agent can make radial marks at any point. If there are n agents then every agent needs to make $n+1$ radial marks on the cake creating $n+1$ pieces of the cake. The allocation procedure will allot one of these pieces to the agent. Thus there will be a total of $(n+1)^2$ marks of the cake. Once agents submit their bid on how the cake will be cut, each agent will be allotted one piece of the cake demarcated by its own marks that will have a worth of $1/(n+1)$ in its opinion. If every agent follows the protocol, then fair allotments are guaranteed for every agent.

Figure 2 shows the interaction diagram for the protocol. As will be shown later, due to inefficiency in the allocation process, there will be leftovers which may not be negligible in the valuation of the agents. In such a case the agents may ask to bid for the leftovers using the same bidding protocol.

Theorem 2. If there are n agents and each agent makes $n+1$ radial marks, creating $n+1$ portions of the circular cake, then our protocol guarantees that each agent will be allotted a piece of the cake, such that the piece was one of the $n+1$ pieces created by the agent itself.

Proof: This proof assumes that the marks are strictly in increasing order, viz. no two points are in exactly the same position. We define two terms: A *k-circle* and a *k-sector*.

k-circle. A *k-circle* consists of a circle that has $k+1$ marks made by each of the k agents demarcating $k+1$ intervals.

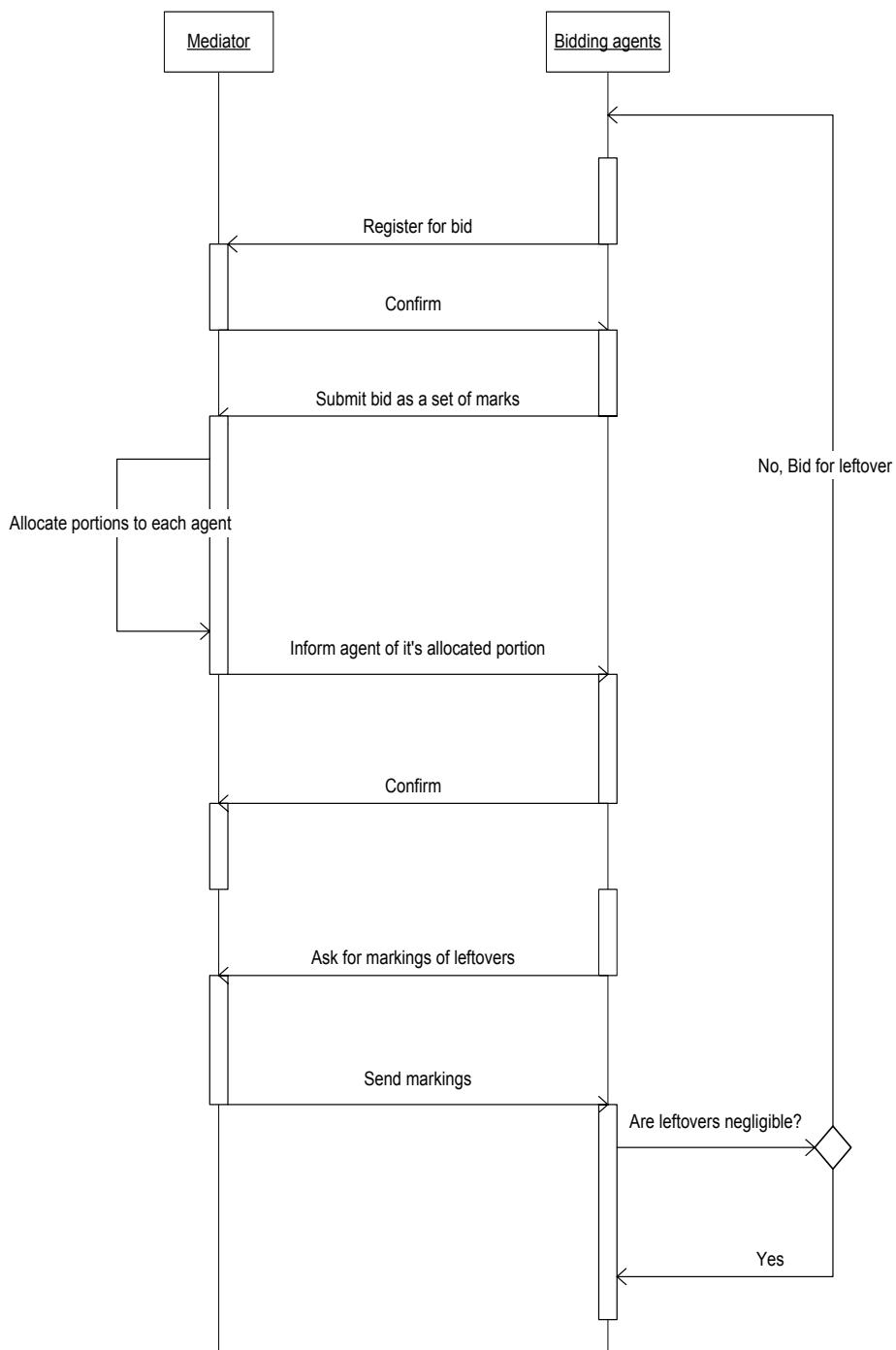
k-sector. A *k-sector* is formed when one or more intervals have been removed from a circle. A *k-sector* contains at least $k+1$ marks demarcating at least k portions for each of the k agents. By this definition, a *k-sector* is also a $(k-1)$ -sector, which is also a $(k-2)$ -sector, etc. A *k-sector* may or may not be a $(k+1)$ -sector however.

When agents mark their intervals, the following possibilities exist for a particular chosen interval.

1. Pure interval, i.e., no other agent's interval intersects this interval
2. Mixed interval
 - 2.1. The current interval partially intersects another interval. It does not completely contain any other interval.
 - 2.2. The current interval contains at least one interval made by at least one agent. In addition, there will definitely be an interval that partially intersects the current interval.

In the case of a *k-circle*, the following events may occur:

1. If the interval allotted was pure, then the other agent's intervals were outside this interval, i.e., the portions demarcated by other agents included the current interval and some extra portions ahead and behind the current interval. Since the current interval was allocated, the corresponding portions of the other agents are destroyed because they can no longer include the currently allocated portion.

**Fig. 2.** Circular cake cutting protocol

2. If the interval allotted was mixed, then there was some overlap with other agents' portions. When this interval is allocated, then it destroys the two intervals of the other agents and they can no longer be allocated to the respective agents.

The above cases show that for a k-circle, allotting an interval will destroy at least one and at most 2 intervals of other agents.

In the case of a k-sector, the following events may occur.

1. If the interval allotted was pure then at most 1 interval of other agents is destroyed.
2. If the interval allotted was mixed then it means other agents interval started after the current interval. Allocating the current interval would destroy at most one interval of the other agents.

The above cases show that in case of a k-sector, allotting an interval will destroy at most one interval of the other agents. We will now show an inductive proof for the allocation procedure.

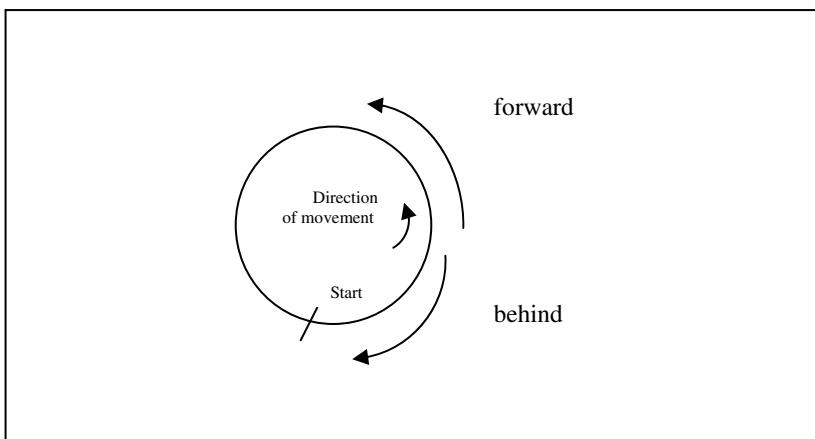


Fig. 3. Conventions used in the proof

Base Case ($n=2$). For 2 agents we will have a 2-circle with each agent making 3 marks and demarcating 3 intervals respectively. Let the agents be A and B. Arbitrarily mark a position on the circumference as a “start” mark. Without losing generality, we choose to move in the counterclockwise direction. In this proof, “moving back” means moving clockwise till one has reached the start mark. Similarly “moving forward” means moving counterclockwise till one has reached the start mark. Find the first mark and assume this mark belongs to agent A. The next mark may or may not be made by A.

1. If the next mark is made by A, this is a pure interval, so allocate this interval to A. Remove all marks to the back of this interval. Remove all marks made by agent A. Thus we will be left with a 1-sector having marks from agent B only. Since none of B's marks have been erased, the 1-sector will have 3 marks demarcating 2 intervals. Allot any of these intervals to B.

2. If the next mark is made by B, A's interval is mixed. Either case 2.1 or 2.2 will occur.
- 2.1. A's interval intersects partially with B. In this case allocate the interval to A. Remove all marks behind the interval. Remove all of A's marks. We will be left with a 1-sector. Since A's interval intersected with B's interval, the previous step will reduce B's marks to 2. Thus the 1-sector will have 2 marks demarcating 1 interval. Allot this interval to B.
 - 2.2. A contains at least one interval of B. Since there are no more agents, such an interval of B is guaranteed to be a pure interval and is allocated to B. Remove all marks behind the interval. Remove all of B's marks. We will be left with a 1-sector. Since A contained B, the 1-sector will have 2 marks demarcating 1 interval. Allot this interval to A.

This proves that A and B can be allocated fair shares, no matter how the marks are arranged. Now let us turn our attention to the general case of n agents.

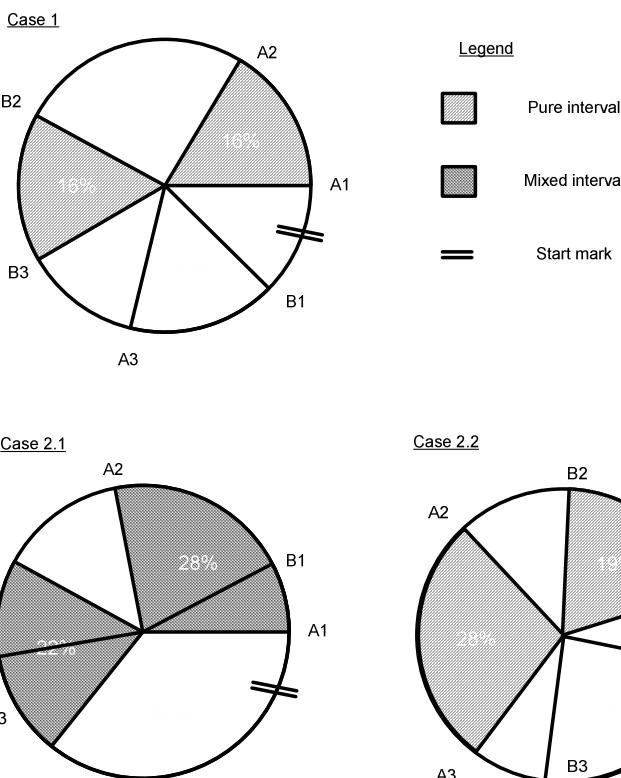


Fig. 4. Allocations for $n=2$ for a circular resource

For any n ($n > 2$). Let us assume that the allocation procedure works for up to k agents. In order for the allocation procedure to work, one of the k agents would have

been allotted an interval from a k -circle. After this a $(k-1)$ -sector would have formed, which was used to allot intervals for the rest of the $k-1$ agents. Hence we can conclude that the allocation procedure works for up to $(k-1)$ -sector. We will show that the procedure works for $k+1$ agents.

For $k+1$ agents we create a $(k+1)$ -circle which will have $k+2$ marks demarcating $k+2$ portions for each of the agents. Find the first mark. Say this mark belongs to agent i . The next mark may or may not be made by i .

1. If next mark is made by i , this is a pure interval, allocate current interval to i . Remove all marks behind this interval. Remove all marks made by agent i . None of the marks made by other agents will be removed as this was a pure interval for agent i . We will be left with a k -sector that has $k+2$ marks demarcating $k+1$ intervals. Read in the next mark. Say it belongs to agent j .
 - 1.1. If the mark after this belongs to j then this is a pure interval which can be allocated to j . Remove all marks made by j . We will be left with a $(k-1)$ -sector that has $k+2$ portions demarcating $k+1$ intervals for each of the $k-1$ agents. Since $k-1$ sector problem has been solved, Hence proved.
 - 1.2. If the mark belongs to some other agent l , then add agent j to list of agents whose mark has already been seen and repeat allocation procedure for the k -sector with agent l 's mark as the first mark. Once an interval has been allocated to an agent, remove all marks of the agent. Remove all marks behind this interval. Other agents may lose at most one mark. Thus we now have a $(k-1)$ -sector problem where $k-1$ agents have at least $k+1$ marks and demarcating k intervals. Hence proved.
2. If the next mark is not made by i , suppose the mark belongs to agent j . Add i to the list of agents whose mark has already been seen. Repeat the allocation procedure with j 's mark as the first mark. At some point we will encounter a mark made by one of the agents already in the list. Say the first such agent is l . The interval demarcated by l will not contain any other agent's interval. It may or may not partially intersect with other agent's intervals.
 - 2.1. If l 's interval does not intersect with any other interval, then allocate the interval to l . Remove all marks made by agent l . We will be left with a k -sector that has $k+2$ marks demarcating $k+1$ intervals for each of the k agents. This k -sector is the same as the one discussed in case 1.1 whose solution has been described. Hence proved.
 - 2.2. If l 's interval partially intersects some other interval, then allocate the interval to l . Remove all marks made by agent l . The previous step will remove at most one mark of an agent. Other agents will have $k+2$ marks and $k+1$ intervals. We will be left with a k -sector having at least $k+1$ marks demarcating k intervals. Read in the next mark. Say it belongs to agent j .
 - 2.2.1. If the mark after this belongs to j then this is a pure interval, which can be allocated to j . Remove all marks made by j . We will be left with a $(k-1)$ -sector having at least $k+1$ marks demarcating k intervals for each of the $k-1$ agents. Since $k-1$ sector problem has been solved, hence proved.
 - 2.2.2. If the mark belongs to some other agent l , then add agent j to the list of agents whose mark has already been seen and repeat allocation procedure for the k -sector with agent l 's mark as the first mark. Once an interval has been allocated to an agent, remove all marks of the agent. Remove all marks behind this interval. Other agents may lose at most

one mark. Thus we now have a $(k-1)$ -sector problem where $k-1$ agents have at least k marks and demarcating $k-1$ intervals, hence proved.

This proves that the allocation procedure works for n agents, for any $n \geq 2$. Next we discuss the important features of this protocol.

4.2 Features

The protocol above has the following features:

- 1) The protocol is fair because each agent gets one of the pieces he demarcated for himself and that piece is worth at least $1/(n+1)$ of the whole cake.
- 2) It is unbiased because there is no mediator bias as seen in the linear version of the algorithm. Each agent can have its own start mark, rather than the mediator choosing the start mark.
- 3) Since each agent makes $n+1$ marks, the space complexity is $O(n^2)$. The protocol may make n comparisons in the first iteration, $n-1$ comparisons in the next, and so on in the worst case scenario. Thus the time complexity is $O(n(n+1)/2)$.
- 4) One feature of this algorithm is that agents need not be ordered. Most other protocols assume that agents agree to order themselves in a certain way, but such an issue can be contentious in itself. For example, in the successive pairs algorithm agents might prefer to come later rather than earlier to avoid the labor of redividing their portions into ever smaller pieces in each iteration.

The algorithm is inefficient, because for n agents, each agent is expected to create $n+1$ intervals. When one of the intervals is allotted to the agent, it is actually getting $1/(n+1)$ of the whole cake. This inefficiency can however be reduced by joining together the wasted pieces and forming a sector. All the agents can then make $n+1$ marks demarcating n intervals in this sector. The sector algorithm can then be applied to allocate portions to each. This can be repeated until the wasted portion is negligible in every agent's opinion. The algorithm is therefore applicable only when resources are infinitely divisible into combinable portions.

Thus by reapplying the sector algorithm a finite number of times, this allocation procedure can get arbitrarily close to the $1/n^{th}$ allocation for each agent. This protocol assumes that the problem domain is infinitely divisible and combinable. In order to give an idea of the various types of problem domains that exist, refer to the following table.

Table 1. Various problem domain types

Problem domain properties	Infinitely divisible	Combinable
A single row of seats in a theatre	--	--
Time scheduling	X	--
Circular cake	X	X

5 Conclusion and Discussion

We have presented a new protocol for allocating linear and circular resources, extending the classic cake cutting problem. This algorithm is applicable for n agents in general. It has been modified specially for the case of a circular resource. The proof

shows that an allotment chosen by the agent itself is guaranteed for each of the n agents. In addition to this, the protocol is fair and unbiased, features that are highly desirable but generally difficult to achieve. However, the first allotment of n pieces is inefficient, because in each agent's opinion only $1/(n+1)^{th}$ portion of the resource is received. To improve efficiency, we reapply the allotment procedure to the wasted portions of the resource. Agents can run the procedure a fixed number of times or until they all agree that the wasted portion is negligible and further division is unnecessary. The current procedure neither demands nor exposes the utility functions of individual agents. Although a mediator may be used to allot the various portions of the resource, the solution of the mediator can be verified, unlike the case of the moving knife solution. Hence disputes can be resolved easily. Also to be noted is that the mediator need not have any features, such as being unbiased. In fact the mediator can be one of the bidding agents. This is possible because the allocation procedure is completely algorithmic and does not depend on the subjectivity of the mediator. If any agent thinks the allocation was unfair, it can re-run the procedure to confirm the validity of the allocation. Unlike many other protocols, such as successive pairs or divide-and-conquer, our protocol does not require an implicit ordering of agents. This avoids any disputes as to which will be the first one to divide the resource. However, the space and time complexity of this procedure is relatively poor. Hence users will have to study the cost versus quality trade-offs to determine which protocol they find most suitable for their resource allocation needs. Among the issues that need to be looked into further are:

1. Improving the space- time complexity of the allocation procedure
2. Since the procedure requires a centralized mediator to run the allocation procedure, a distributed version of the allocation procedure could ease the load of the computation on the mediator.

References

- [1] Rosenchein, J.S., and Zlotkin, G. 1994. Rules of Encounter. London, England.: MIT Press.
- [2] Davis, R. and Smith, R. January 1983. Negotiation as a Metaphor for Distributed Problem Solving. Artificial Intelligence 20(1): 63-109
- [3] Steinhause, H. 1948. The problem of fair division. Econometrica 16:101-104
- [4] Brams, S. J., and Taylor, A. D. 1996. Fair Division: From cake-cutting to dispute resolution. Cambridge,UK.: Cambridge University Press.
- [5] Biswas, A. and Sen, S. More than envy-free. In the Working Papers of the AAAI-99 Workshop on Negotiation: Settling Conflicts and Identifying Opportunities, 44-49. Menlo Park, CA: AAAI Press.
- [6] Tasnadi, A. November 2003. A new proportional procedure for the n-person cake-cutting problem. Economics Bulletin 4:1-3.
- [7] Robertson, J., and Webb, W. 1998. Cake-Cutting Algorithms: Be Fair if You Can. Natick, MA: A.K.Peters.
- [8] Huhns, M.N. and Malhotra, A.K. July 1999. Negotiating for Goods and Services. IEEE Internet Computing, 3(4): 97-99.
- [9] Stewart, I. December 1998. Mathematical Recreations: Your Half 's Bigger Than My Half!. Scientific American 112-114.

QoS-Based Service Selection and Ranking with Trust and Reputation Management^{*}

Le-Hung Vu, Manfred Hauswirth, and Karl Aberer

School of Computer and Communication Sciences,
Ecole Polytechnique Fédérale de Lausanne (EPFL),
CH-1015 Lausanne, Switzerland

{lehung.vu, manfred.hauswirth, karl.aberer}@epfl.ch

Abstract. QoS-based service selection mechanisms will play an essential role in service-oriented architectures, as e-Business applications want to use services that most accurately meet their requirements. Standard approaches in this field typically are based on the prediction of services' performance from the quality advertised by providers as well as from feedback of users on the actual levels of QoS delivered to them. The key issue in this setting is to detect and deal with false ratings by dishonest providers and users, which has only received limited attention so far. In this paper, we present a new QoS-based semantic web service selection and ranking solution with the application of a trust and reputation management method to address this problem. We will give a formal description of our approach and validate it with experiments which demonstrate that our solution yields high-quality results under various realistic cheating behaviors.

1 Introduction

One key issue in the Semantic Web Service area is to discover the most relevant services meeting the functional requirements of users. Equally important, e-Business applications also would like to discover services which best meet their requirements in terms of QoS, i.e., performance, throughput, reliability, availability, trust, etc. Thus QoS-based web service selection and ranking mechanisms will play an essential role in service-oriented architectures, especially when the semantic matchmaking process returns lots of services with comparable functionalities.

In this paper we present a QoS-based web service selection and ranking approach which uses trust and reputation evaluation techniques to predict the

* The work presented in this paper was (partly) carried out in the framework of the EPFL Center for Global Computing and was supported by the Swiss National Funding Agency OFES as part of the European project DIP (Data, Information, and Process Integration with Semantic Web Services) No 507483. Le-Hung Vu is supported by a scholarship of the Swiss federal government for foreign students. We also thank Zoran Despotovic, Amit Lakhani and the anonymous reviewers for their carefully reading and commenting this paper.

future quality of a service. Our work is based on requirements from a real-world case study on virtual Internet service providers (ViSP) from an industrial partner in one of our projects¹. In a nutshell, the idea behind the ViSP business model is that Internet Service Providers (ISPs) describe their services as semantic web services, including QoS such as availability, acceptable response time, throughput, etc., and a company interested in providing Internet access, i.e., becoming a ViSP, can look for its desired combination of services taking into account its QoS and budgeting requirements, and combine them into a new (virtual) product which can then be sold on the market. This business model already exists, but is supported completely manually. Since many ISPs can provide the basic services at different levels and with various pricing models, dishonest providers could claim arbitrary QoS properties to attract interested parties. The standard way to prevent this is to allow users to evaluate a service and provide feedbacks. However, the feedback mechanism has to ensure that false ratings, for example, badmouthing about a competitor's service or pushing one's own rating level by fake reports or collusion with other malicious parties, can be detected and dealt with. Consequently, a good service discovery engine would have to take into account not only the functional suitability of services but also their prospective quality offered to end-users by assessing the trustworthiness of both providers and consumer reports. According to several empirical studies [1, 2], the issue of evaluating the credibility of user reports is one of the essential problems to be solved in the e-Business application area.

We develop the QoS-based service selection algorithm under two basic assumptions which are very reasonable and realistic in e-Business settings: First, we assume *probabilistic behavior of services and users*. This implies that the differences between the real quality conformance which users obtained and the QoS values they report follow certain probability distributions. These differences vary depending on whether users are honest or cheating as well as on the level of changes in their behaviors. Secondly, we presume that *there exist a few trusted third parties*. These well-known trusted agents always produce credible QoS reports and are used as trustworthy information sources to evaluate the behaviors of the other users. In reality, companies managing the service searching engines can deploy special applications themselves to obtain their own experience on QoS of some specific web services. Alternatively, they can also hire third party companies to do these QoS monitoring tasks for them. In contrast to other models [3, 4, 5, 6, 7] we do not deploy these agents to collect performance data of all available services in the registry. Instead, we only use a small number of them to monitor QoS of some selected services because such special agents are usually costly to setup and maintain.

The QoS-based service selection and ranking algorithm we describe in this paper is a part of our overall distributed service discovery approach [8]. During the service discovery phase, after the functional matchmaking at a specific registry, we would obtain a list of web services with similar functionalities from the matchmaking of our framework, i.e., the services fulfilling all user's functional

¹ DIP Integrated Project, <http://dip.semanticweb.org/>

requirements. We need to select and rank these services based on their predicted QoS values, taking into consideration the explicit quality requirements of users in the queries. The output of the selection and ranking algorithm is the list of web services fulfilling all quality requirements of a user, ordered by their prospective levels of satisfaction of the given QoS criteria. So as to perform this selection and ranking accurately, we collect user reports on QoS of all services over time to predict their future quality. This prediction is also based on the quality promised by the service providers as well as takes into consideration trust and reputation issues.

The major contribution of our work is a new QoS-based web service selection and ranking approach which is expected to be accurate, efficient and reliable. First, we have taken into account the issue of trust and reputation management adequately when predicting service performance and ranking web services based on their past QoS data. Experimental results have shown that the newly proposed service selection and ranking algorithm yields very good results under various cheating behaviors of users, which is mainly due to the fact that *our use of trusted third parties observing a relatively small fraction of services can greatly improve the detection of dishonest behavior even in extremely hostile environments*. This is particularly important as without cheating detection, service providers will be likely to generate lots of false reports in order to obtain higher ranks in the searching results, thereby having higher probability to be selected by clients and gain more profits. Second, our algorithm is semantic-enabled by offering support for the semantic similarity among QoS concepts advertised by providers and the ones required by users. This allows the QoS-based service selection process to work more flexibly and produce more accurate results. Third, we adapt the idea of Srinivasan et al [9] to pre-compute all matching information between QoS capabilities of published services and possible QoS requirements of users to avoid time-consuming reasoning and to minimize the searching costs.

The rest of this paper is organized as follows. First, we briefly mention the related work in section 2. Section 3 presents our trust and reputation management model in a Web Service Discovery scenario. Our QoS-based service selection and ranking approach is described in detail in section 4. We discuss various experimental results in section 5 and conclude the paper in section 6.

2 Related Work

Although the traditional UDDI standard² does not refer to QoS for web services, many proposals have been devised to extend the original model and describe web services' quality capabilities, e.g., QML, WSLA and WSOL [10]. The issue of trust and reputation management in Internet-based applications has also been a well-studied problem [1, 2].

The UX architecture [11] suggests using dedicated servers to collect feedback of consumers and then predict the future performance of published services. [12] proposes an extended implementation of the UDDI standard to store QoS

² Latest UDDI Version (3.0.2), <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>

data submitted by either service providers or consumers and suggests a special query language (SWSQL) to manipulate, publish, rate and select these data from repository. Kalepu et al [13] evaluate the reputation of a service as a function of three factors: ratings made by users, service quality compliance, and its verity, i.e., the changes of service quality conformance over time. However, these solutions do not take into account the trustworthiness of QoS reports produced by users, which is important to assure the accuracy of the QoS-based web service selection and ranking results. [14] rates services computationally in terms of their quality performance from QoS information provided by monitoring services and users. The authors also employ a simple approach of reputation management by identifying every requester to avoid report flooding. In [15], services are allowed to vote for quality and trustworthiness of each other and the service discovery engine utilizes the concept of distinct sum count in sketch theory to compute the QoS reputation for every service. However, these reputation management techniques are still simple and not robust against various cheating behaviors, e.g., collusion among providers and reporters with varying actions over time. Consequently, the quality of the searching results of those discovery systems will not be assured if there are lots of colluding, dishonest users trying to boost the quality of their own services and badmouth about the other ones. [16] suggests augmenting service clients with QoS monitoring, analysis, and selection capabilities, which is a bit unrealistic as each service consumer would have to take the heavy processing role of a discovery and reputation system. Other solutions [3, 4, 5, 6, 7] use third-party service brokers or specialized monitoring agents to collect performance of all available services in registries, which would be expensive in reality. Though [7] also raises the issue of accountability of Web Service Agent Proxies in their system, the evaluation of trust and reputation for these agents is still an open problem.

Our QoS provisioning model is grounded on previous work of [3, 4, 11, 13, 14]. The trust and reputation management of our work is most similar to that of [17, 18] but we employ the idea of distrust propagation more accurately by observing that trust information from a user report can also be used to reveal dishonesty of other reporters and by allowing this distrust to be propagated to similar ones. Other ideas of the trust and reputation management method are based on [19, 20, 21].

3 A Trust and Reputation Management Model for QoS-Based Service Discovery

The interaction among agents in our system is represented in Fig. 1 where S_0, \dots, S_n are web services, U_0, \dots, U_m are service users, RP_0, \dots, RP_k are service registries in a P2P-based repository network, based on our P-Grid structured overlay [22]. T_0, \dots, T_r are the trusted QoS monitoring agents from which we will collect trustworthy QoS data to use in our algorithm.

After U_j 's perception of a normalized QoS conformance value x_{ij} (a real number from Definition 1 in the following section) from a service S_i , it may report

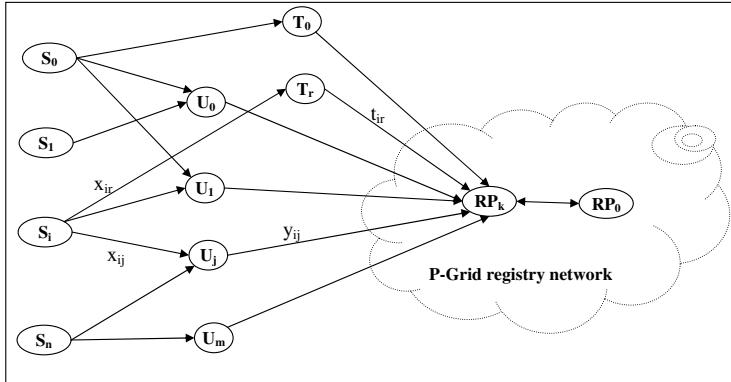


Fig. 1. Interaction among agents in a QoS-based service discovery scenario

the value y_{ij} to the registry RP_k , which manages the description information of S_i . This registry peer will collect users' quality feedbacks on all of its managed web services to predict their future performance and support the QoS-enabled service discovery based on these data. Note that a user generally reports a vector of values representing its perception of various quality parameters from a service. Also, x_{ij} s and y_{ij} s are QoS conformance values, which already take into account the quality values advertised by providers (see Definition 1 in the next section). Since the prediction of QoS in our work mostly depends on reports of users to evaluate service performance, the cheating behaviors of providers are not explicitly mentioned henceforth. In this paper, we only consider the selection and ranking of services with reputation management in one registry peer. The study of interaction among different registries is subject to future work.

Given the above interaction model, we can make a number of observations. An honest user will report $y_{ij} = x_{ij}$ in most cases (in reality, they may not report if not willing to do so). On the other hand, a dishonest reporter who wants to boost the performance of S_i will submit a value $y_{ij} > x_{ij}$. Similarly, cheating reports generated by S_i 's competitors will report $y_{ij} < x_{ij}$ to lower its QoS reputation. In addition, (colluding) liars may sometimes provide honest reports and behave dishonestly in other cases [2]. Accordingly, we presume that the differences between y_{ij} s and x_{ij} s follow certain distribution types which *expected values* depend on the behavior of the user, i.e., are equal to 0.0 for honest users and different from 0.0 in the case of liars. We use this assumption since in general dishonest users are likely to change their actions over time in order to hide their cheating behaviors with *occasionally* credible reports. Moreover, as the quality parameter values of a service really depend on many environmental factors, even trusted agents and honest users may obtain and report those values a little different to each other when talking about the same service. However, the expected value of trustworthy reports on QoS of S_i , e.g., the average of corresponding credible y_{ij} values, will reflect its real quality capability. In our opinion, the expected values of these distributions, e.g., means, normally reveal

the behaviors of users, whereas other parameters, e.g., standard deviations, will represent the uncertainty in actions of users, either accidentally or by purpose. Our goal is not only to evaluate whether a user is honest but also to compute the expected conformance values from the reports of the most honest users, e.g., the average of values y_{ij} s by the most credible reporters, from which we will predict the future quality of S_i s.

4 QoS-Based Service Selection and Ranking

Our QoS-enabled distributed service discovery framework is presented in detail in [8]. Quality properties of web services are described by concepts from a QoS ontology and then embedded into service description files using techniques suggested by WS-QoS [3] and Ran [4]. The value of a quality parameter of a web service is supposed to be *normalized* to a non-negative real-valued number regarding service-specific and call-specific context information where *higher normalized values represent higher levels of service performance*. We are aware that the issue of normalizing the values of various quality attributes is complicated, but this is out of the scope of our current study. However, with most frequently used QoS concepts, e.g., reliability, execution-time, response-time, availability, etc., the answer is well-defined and straight-forward. For instance, a web service with a normalized QoS parameter value for reliability of 0.90 will be considered as more reliable to another one with a normalized reliability value of 0.50. In this case the normalized reliability is measured as its degree of being capable of maintaining the service and service quality over a time period T . The ontology language we use to describe service semantics and to define the QoS ontology is WSMO³, but other models, e.g., OWL-S⁴, would also be applicable. For experimental evaluations, we have developed a simple QoS ontology for the VISP use case including the most relevant quality parameters for many applications, i.e., availability, reliability, execution time, etc. We currently assume that users and providers share a common ontology to describe various QoS concepts. However, this could be relaxed with the help of many existing ontology mapping frameworks.

4.1 Predicting Service Performance

In order to predict the quality of a web service S_i , we collect all QoS feedbacks on its performance over a time period W and use a *real-valued time series forecasting technique* to predict its future quality conformance from past data. To understand the concepts in our algorithms we start with two fundamental definitions.

Definition 1. The quality conformance value c_{ij}^k of a service S_i in providing a quality attribute q_{ij} at time k is defined as $c_{ij}^k = \frac{d_{ij}^k - p_{ij}^k}{p_{ij}^k}$ where d_{ij}^k is the

³ <http://www.wmso.org/>

⁴ <http://www.daml.org/services/owl-s/>

normalized value of q_{ij} that S_i actually delivered to a specific user at time k and p_{ij}^k is the corresponding normalized QoS value promised by S_i 's provider at that time.

Definition 2. A user QoS report R , either by a normal service consumer or by a trusted monitoring agent, is a vector $\{u, S_i, t, L\}$, where u is the identifier of the user that produced this report, S_i is the corresponding web service, t is the timestamp of the report and L is a quality conformance vector of $\{q_{ij}, c_{ij}^t\}$ pair values, with q_{ij} being a QoS attribute offered by S_i and c_{ij}^t being q_{ij} 's quality conformance that S_i provides to this user at time t .

In order to filter out as much dishonest reports as possible and to take only the most credible ones in the QoS predicting process, we apply our trust and reputation management techniques comprising of two steps: a report preprocessing and a report clustering phase. The first phase evaluates the credibility of collected user reports by applying a trust-and-distrust propagation approach, which relies on some initial trusted reports produced by special monitoring agents. We consider two QoS reports as *comparable* if they are related to the same service during a specific time interval δ_t and as *incomparable* otherwise. Generally, we can set this δ_t as big as the length of the period during which the corresponding service provider does not change the promised quality values of this service. Two *comparable* QoS reports are considered to be *similar* if the squared Euclidean distance between their conformance vectors is less than a specific threshold. On the contrary, they are regarded as *dissimilar* if this distance is greater than another threshold value.

The report preprocessing step is done according to Algorithm 1. n_{ch} and n_{h1}, n_{h2} ($n_{h1} < n_{h2}$) are threshold values to estimate a user as cheating or honest regarding to the similarity of its reports to other cheating/honest ones (line 9, 17 and 18). N and T represent for how long and how frequent a user stay in and submit QoS reports to the system. The values of $n_{h1}, n_{h2}, n_{ch}, N$ and T are design parameters to be chosen depending on properties of the collected reports after running the algorithm several times. Generally, higher values of these parameters stand for higher level of caution when estimating behaviors of users regarding current evidences against them.

After finishing the preprocessing phase, we can identify a certain number of cheaters and honest users. However, this trust-distrust propagation phase may not be able to evaluate the credibility of all reports in case the user communities of certain services are isolated from other communities as well as if we set the values of n_{h1}, n_{h2} and n_{ch} too high in Algorithm 1.

Therefore, in the next step we have to estimate the trustworthiness of the remaining reports of which credibility has not been evaluated. To achieve this, we reason that colluding cheating users will cooperate with each other in order to influence the system with their dishonest feedbacks. As a result, users within each group will produce similar values and naturally form different clusters of reports. Thus it is possible to apply data-mining techniques in this situation to discover various existing clusters of reports related to those user groups. In our work, we apply the *convex k-mean clustering algorithm* on each set of QoS

Algorithm 1. QoSReportsPreprocessing()

```

1: all trusted agents are marked as honest users;
2: all reports of trusted agents are marked honest;
3: repeat
4:   all unmarked reports of each cheating user are marked cheating;
5:   for each unmarked report do
6:     if this report is dissimilar from an honest report then mark it cheating;
7:     if this report is similar with a cheating report then mark it cheating;
8:   end for
9:   users with at least  $n_{ch}$  reports similar with cheating ones are marked cheating;
10:  users with at least  $N$  reports in at least  $T$  different time are marked stable;
11: until there is no new cheating user discovered;
12: repeat
13:   all unmarked reports of each honest user are marked as honest;
14:   for each unmarked report and each report marked cheating do
15:     if this report is similar with an honest report then mark it honest;
16:   end for
17:   unmarked users with at least  $n_{h1}$  reports similar with honest ones are marked honest;
18:   users marked as cheating and having at least  $n_{h2}$  reports similar with honest ones are re-marked honest;
19: until there is no new honest user discovered;

```

reports related to a service during the time interval δ_t with the following metrics: The distance between two *comparable* reports is defined as the Euclidean squared distance between two corresponding quality conformance vectors and the distance between two *incomparable* ones is assigned a large enough value so that these reports will belong to different clusters.

After the trust and reputation evaluation in the two above phases, for each service S_i , we will have a list G_i of groups of reports on QoS of S_i over time. Generally, G_i includes the groups containing those reports that were previously marked as honest/cheating during the trust-distrust propagation phase, as well as other clusters of reports obtained after the clustering step. We will assign the credibility w_i^g of a report group $g_i \in G_i$ as follows. Firstly, we filter out all dishonest ratings by assign $w_i^g = 0.0$ for all groups of reports marked as cheating during the trust-distrust propagation. If there exists the group g_i^0 of reports previously marked honest during that step, we assign $w_i^{g_0} = 1.0$ whereas letting $w_i^g = 0.0$ for the remaining groups. Otherwise, we try to compute w_i^g so that this value would be proportional to the probability that the group g_i is trustworthy among all of the others. Our heuristic is to assign higher weight to clusters which are populated more densely, having bigger size and with more stable users. This assumption is reasonable, as the reports of independent cheaters are likely to be scattered, and in the case liars cooperate with each other to cheat the system, the size of their corresponding clusters will not exceed those consisting only of honest reports as it would be too costly to dominate the system with numerous and clustered dishonest ratings. Even if dishonest providers try to produce lots of

more condense reports so that they could get high influences to the final ranking results at any cost, these values will be separated from honestly reported values and therefore are likely to be discovered during the distrust-propagation phase (line 3 to line 11 in Algorithm 1), provided we have enough trustworthy reports to use. Specifically, w_i^g could be estimated based on the following information: the number of users in the cluster g_i ($size_i^g$), the number of all users producing reports in all clusters of G_i ($allusers_i$), the number of stable users in this cluster ($stable_i^g$), the total number of stable users in all clusters of G_i ($allstable_i$), as well as the average distance d_i^g from the member reports of cluster g_i to its centroid values. Based on our model in section 3, the credibility of one report would depend on the distance between its conformance vector and that of an honest report. Therefore, the similarity among credibility of different reports in one cluster g_i would be inversely proportional to its d_i^g value. Furthermore, a report in g_i would be honest in two cases: (1) it is submitted by a *stable and honest* user; (2) it is produced by an *unstable and honest* user. Let P_{stbl} and P_{unstbl} be the probability that this report is of a stable user and of an unstable user, respectively, and let P_{stblcr} and $P_{unstblcr}$ be the probability that stable and unstable users report credibly, then we have $w_i^g = \frac{C}{d_i^g} \cdot (P_{stbl} \cdot P_{stblcr} + P_{unstbl} \cdot P_{unstblcr})$, where $P_{stbl} = \frac{stable_i^g}{size_i^g}$ and $P_{unstbl} = 1 - P_{stbl}$. P_{stblcr} and $P_{unstblcr}$ could be estimated by comparing reports of trusted agents with those of sample stable/unstable users to derive an appropriate value at the whole network level. The value of C represents our belief in the relation between the density of a report cluster and the credibility of its members, which is considered as parameters of the reputation system and to be set by experience.

The future quality conformance \hat{C}_{ij} of a service S_i in providing a QoS attribute q_{ij} is predicted using a linear regression method, thus we have: $\hat{C}_{ij} = LinearRegression(\bar{C}_{ij}^t)$, $t \in \{0, \delta_t, \dots, (W-1).\delta_t\}$, where \bar{C}_{ij}^t is the evaluated QoS conformance value of the quality parameter q_{ij} at time t . We compute \bar{C}_{ij}^t as the average of conformance values reported by various user groups in the system at that specific time point, using the evaluated credibility w_i^g 's as weights in the computation. In other word, $\bar{C}_{ij}^t = \frac{\sum_{g_i \in G_i} w_i^g C_{ij}^t}{\sum_{g_i \in G_i} w_i^g}$ where C_{ij}^t is the mean of conformances of a report group g_i on S_i 's quality attribute q_{ij} at time t , i.e., a centroid value of a cluster/group of reports produced after the trust and reputation evaluation phase. Regarding the probabilistic behavior of users and services as in section 3, we consider \hat{C}_{ij} as an approximate estimate of the expected value of S_i 's QoS conformance in providing quality attribute q_{ij} to users.

4.2 QoS-Based Service Selection and Ranking

We define QoS requirements in a user query as a vector Q of triples $\{q_j, n_j, v_j\}$ where q_j represents for the required QoS attribute, n_j is the level of importance of this quality attribute to the user and v_j is the minimal delivered QoS value that this user requires. To rank services according to its prospective level of satisfying user's QoS requirements, we utilize the Simple Additive Weighting

Algorithm 2. QoSSelectionRanking(ServiceList L, ServiceQuery Q)

```

1: Derive the list of QoS requirements in Q:  $L_q = \{[q_1, n_1, v_1], \dots, [q_s, n_s, v_s]\}$ 
2: Initialize  $Score[S_{ij}] = 0.0$  for all services  $S_{ij} \in L$ ;
3: for each quality concept  $q_j \in L_q$  do
4:   for each service  $S_{ij} \in L$  do
5:     Search the list  $L_{qos}$  of  $q_j$  for  $S_{ij}$ ;
6:     if  $S_{ij}$  is found then
7:        $Score[S_{ij}] = Score[S_{ij}] + \frac{n_j \cdot w_{ij}}{\sum n_j} (\frac{\hat{d}_{ij} - v_j}{v_j})$ ;
8:     else
9:       Remove  $S_{ij}$  from  $L$ ;
10:    end if
11:   end for
12: end for
13: Return the list  $L$  sorted in descending order by  $Score[S_{ij}]$  s;

```

method, which produces ranking results very close to those of more sophisticated Decision Making techniques [5]. Thus, the QoS rank of a service S_i in fulfilling all quality criteria depends on the weighted sum $T_i = \frac{\sum_{q_j \in Q} n_j \cdot P_{ij}}{\sum_{q_j \in Q} n_j}$ in which $P_{ij} = w_{ij} \cdot \widehat{nd}_{ij}$ represents the capability of S_i in providing the QoS concept q_{ij} for users at the query time. The value $\widehat{nd}_{ij} = \frac{\hat{d}_{ij} - v_j}{v_j}$ evaluates the difference between the QoS value \hat{d}_{ij} of the quality attribute q_{ij} that S_i is able to offer to its users according to the prediction and the corresponding value v_j of q_j required by the service query. In combination with Definition 1, we can compute $\widehat{nd}_{ij} = \frac{(1+p_{ij}) \cdot \hat{C}_{ij} - v_j}{v_j}$, where \hat{C}_{ij} is the predicted QoS conformance value of quality attribute q_{ij} and p_{ij} is the corresponding QoS value promised by provider of S_i at current time. w_{ij} is a weight proportional to the semantic similarity m_{ij} between q_{ij} and the QoS ontology concept q_j required by the user, i.e., the degree of match as in [23]. In other word, we will give higher ranks for services which offer the most accurate QoS concepts at the higher levels compared to the ones required by users. In our program we simple use the following definition to evaluate w_{ij} :

$$w_{ij} = \begin{cases} 1.0 & \text{if } m_{ij} = \text{exact}; \text{(i.e., } q_{ij} \text{ is equivalent to } q_j) \\ 0.5 & \text{if } m_{ij} = \text{pluggin}; \text{(i.e., } q_{ij} \text{ is more general than } q_j) \\ 0.0 & \text{if } m_{ij} \in \{\text{subsume, failed}\}; \text{(i.e., otherwise)} \end{cases} \quad (1)$$

In order to accelerate the selection of only services fulfilling all required QoS parameters, we use the idea of Srinivasan et al [9] to avoid the time-consuming semantic reasoning step. Specifically, we use a *QoS matching table* to store the matching information for all frequently accessed QoS attributes. With each QoS attribute q_j in this table, we have a list L_{qos_j} of records $\{S_{ij}, w_{ij}, \hat{d}_{ij}\}$ where w_{ij} , \hat{d}_{ij} are computed as above and S_{ij} identifies a service having certain support for q_j . Given the list L of services with similar functionalities, the discovery engine performs the QoS-based service selection and ranking process as in Algorithm 2.

5 Experimental Results and Discussions

To evaluate the selection and ranking algorithm, we have implemented it as a QoS support module in a registry peer of our distributed discovery framework [8] and studied its effectiveness under various settings. The service selection and ranking is performed on three representative quality parameters, namely *availability*, *reliability* and *execution-time* taken from the VISP case study.

We observed the dependency between the quality of selection and ranking results and other factors, such as the percentage of trusted users and reports, the rate of cheating users in the user society and the various behaviors of users. Specifically, we evaluated the effectiveness of the service discovery by studying the differences in the quality of results when running evaluations in four different settings: In the *ideal* case the discovery engine has complete knowledge, such that it knows all correct QoS conformance values of all published services in the system over a time window W and performs the selection and ranking of services from this ideal data set. In the *realistic* case, we ran our algorithm with the application of trust and management techniques to filter out incredible reports and to evaluate the credibility for the others. The *optimistic* case corresponds to the QoS-based discovery of services without regarding to trust and reputation issues, i.e., the system simply uses the average of all reported conformance values to predict services' performance and to perform the QoS ranking. The *naive* case corresponds to the selection of services based only on their QoS values promised by the providers, i.e., the system trusts all providers completely. Our goal was to show that the obtained results of the QoS-based service discovery process are more accurate and reliable in the realistic case with various cheating behaviors of users, they would be much worse in the optimistic case, and the worst with the naive method thus clearly showing the contribution of our approach.

The quality of selection results produced by our algorithm can be measured by four parameters, namely *recall*, *precision*, *R-precision* and *Top-K precision*, of which the *R-precision* is the most important quality parameter as recognized by the Information Retrieval community. In our settings, these parameters generally represent the fraction of services that are most relevant to a user among all returned services in terms of their *real* QoS capabilities. Apparently, the results would be the best in the ideal case, i.e., its recall, precision, R-precision and Top-K precision parameters are all equal to 1.0. Therefore, we use the results of the ideal case as a reference to compare with the quality parameters in the other three situations. Due to space limitations we only show the *R-precision* values as the most representative experimental results in this section. We also measured the other parameters as well as computed the absolute QoS ranks of returned services using weighting Spearman's footnote method and had similar results.

We prepared our experiments with the probabilistic assumption on the behavior of service providers and consumers. In this paper we only present the experiments with Gaussian (normal) distributions. The extension to other probabilistic distributions is subject to future work. The society of service consumers was modeled as a collection of different types of users. As mentioned

in section 3, honest users and trusted agents would report values with the difference $D_h \sim Normal(0.0, \sigma_h^2)$ to the real QoS conformance capabilities of services. On the contrary, cheaters would report values with the difference $D_c \sim Normal(M_c, \sigma_c^2)$ to the real quality conformances that they had obtained. The values of the mean M_c varied according to the purpose of the cheaters, i.e., advertise or badmouth a service. The values of σ_c represented the variation in reported values of users among different quality attributes of different services. Users with higher values of σ_c had higher levels of inconsistency in their behaviors and therefore were harder to be detected. We further divided these liars into three sub-types: *badmouthing users* who mostly reported badly about services of their competitors, *advertising users* who usually exaggerated performance of their own services and *uncertain users* with indeterministic actions and who might act as advertising, badmouthing or even as honest users. We set the values of M_c for each type of cheaters in the most pessimistic situation, making our testing environment be very hostile. Specifically, cheaters had their corresponding M_c s set to high/low enough values such that badmouthing users would succeed in pushing services of their competitors out of the selection results and advertising users would be able to raise the QoS ranks of their own services, provided that their reports had been taken into the predicting process of the QoS-based service discovery engine. This setting is realistic because in business, companies generally have knowledge of the base requirements of their users as well as owning certain statistics of their competitors' capabilities. More complicatedly, uncertain users had their M_c s values belonging to N_c specific values each of which was a randomized real value, with N_c was the number of groups of cheaters with varied behaviors. These types of liars would be harder to be detected since their M_c values were uniformly distributed around 0.0. Though they did not contribute directly to the boosting and lowering the reputation of certain services, their reports were not so dissimilar from honest reports in most cases and therefore they would act as good recommenders for other badmouthing/advertising guys. To assure the scalability of the approach, we also tested it with an increasing number of services, users and QoS reports while keeping the percentage of user types and other parameters the same. The remaining experiments were run in a society of 1000 users which produced a total of 50000 QoS reports on 200 services during a time window of length $W = 5$ and $\delta_t = 1$. The results of each experiment were averaged over 10 runs.

As a first question, we wanted to study the effects of the trusted reports on the quality of results in the realistic case. Specifically, we wanted to observe the effects of the percentage of the services monitored by trusted agents $F_{special}$ to the results of our QoS-based service selection and ranking algorithm expressed by *R-Precision* values. We increased $F_{special}$ from 1.0% to 10.0%. The percentage of trusted users/reports was also increased from 0.1% to 1.0% with the increment of 0.1% each step as well. The results of this experiment are shown in Fig. 2.

Correspondingly, Fig. 3 shows the percentage of cheating and honest reports correctly identified during the report preprocessing phase with our trust-distrust propagation method.

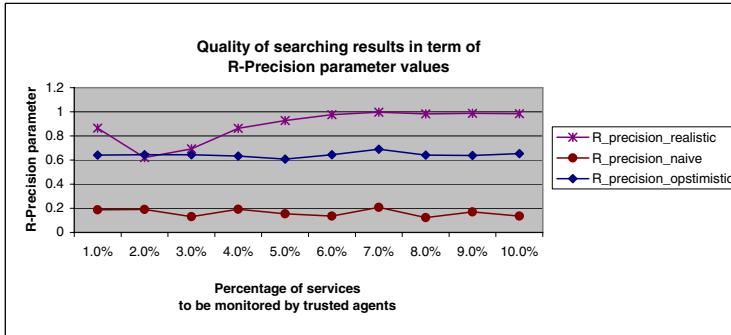


Fig. 2. $F_{special}$ vs. R -Precision

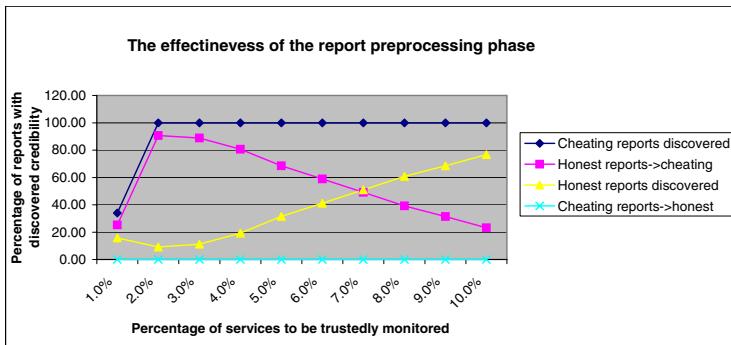


Fig. 3. $F_{special}$ vs. Correctness of the report preprocessing phase

In this experiment, we assumed a very high number of cheaters (74.0% of the total users) consisting of badmouthing users, advertisers and five different groups of uncertain users. We also did various experiments to study the effects of σ_c 's and σ_h 's values and found that our algorithm performed very well with different settings of σ_c provided the quality of the honest user population was good, i.e., σ_h was low. Given the fact that the QoS conformance values in our simulation are in the interval $[-1.0, 1.0]$, we kept the standard deviations of cheating reports very high ($\sigma_c = 0.5$) and those of trusted and honest users at an acceptably low level ($\sigma_h = 0.01$). With the increase of $F_{special}$, we could correctly discover almost all cheating reports and an increasing percentage of honest reports. Accordingly, the quality of the results was significantly increased as well. The clustering phase was actually not performed with $F_{special} > 1.0\%$ because above this threshold, using only the trust-distrust propagation was enough to evaluate the credibility of all reports. Although a lot of honest reports were wrongly identified as cheating, which was due to our cautious approach in estimating report credibility, the quality of the results was always very good if $F_{special}$ was kept high enough

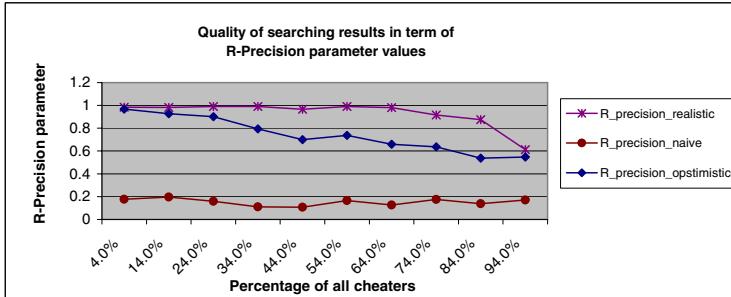


Fig. 4. $F_{cheating}$ vs. R -Precision

(about 5.0%). The results were the worst with $F_{special}$ around 2.0% and 3.0%. In these cases, as the trust-propagation was not effective and did not identify enough honest reports, we had to (partly) use the quality value advertised by providers instead. When $F_{special}$ was very small (1.0%), the result was still acceptable (R -Precision = 0.8), since in this situation we could actually combine the trust-propagation and the report clustering phase together. i.e., there were enough reports with unknown credibility after the preprocessing of reports such that the clustering step had enough input data to process. As a whole, the result of our algorithm with the trust and reputation management scheme in place is much better than that of the optimistic and the naive cases, as we expected.

Next, we studied the effects of the fraction of cheaters to the quality of results. We increased the total percentage of all types of cheating users ($F_{cheating}$), which consists of badmouthing, advertising and uncertain users, from 4.0% to 94.0% in increment of 10% each step. More specifically, we raised the percentage of badmouthing and advertising users/reports, from 3.33% to 78.33% while generating five different groups of uncertain users with corresponding percentage of dishonest reports increased from 0.67% to 15.67%. This setting represents the realistic situation when there are various types of dishonest providers colluding with the generated cheating users to boost the reputation of certain services and badmouth other ones, which could be considered as the most important case where there are various types of users with changing behaviors. The results for this experiment are shown in Fig. 4. We kept the percentage of trusted reports at 0.5% and let $F_{special} = 5.0\%$, as an acceptable fraction collected from observations in the first experiment. The standard deviations of of cheating and honest reports were kept at $\sigma_c = 0.5$ and $\sigma_h = 0.01$ respectively.

With the reduction of honest users and the corresponding increase of $F_{cheating}$, the values of the R -Precision parameter were also reduced. However, the quality of the results in the realistic case was always much better than that of the optimistic case and the naive case. Even when the fraction of cheaters $F_{cheating}$ was very high (0.84), the R -Precision parameter value in the realistic case was still acceptable (higher than 0.8). On the other hand, the quality of results without taking into account trust and reputation management issues, i.e., the optimistic and the naive case, dropped dramatically in hostile settings.

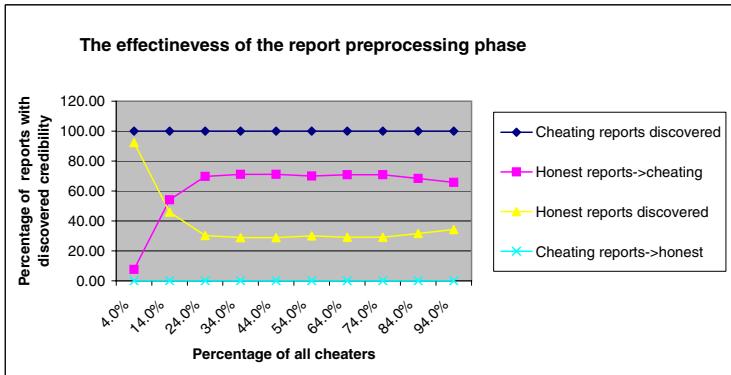


Fig. 5. $F_{cheating}$ vs. Correctness of the report preprocessing phase

This phenomenon was due to the fact that in a society with a very high cheating rate, our trust and reputation evaluation mechanism could discover and filter out almost all incredible reports, as shown in Fig. 5.

From these experiments we can draw a number of conclusions. Regarding efficiency and effectiveness, our selection and ranking approach exhibits the following properties: As the trust and reputation evaluation phase uses social network-based analysis (trust-distrust propagation) and data-mining (report clustering) methods, it requires high-computational cost. Fortunately, in our case, the computation involves mainly local processing in one registry and thus does not require much communication overheads. Additionally, it could be done *off-line* on a periodical basis and therefore will not affect much to the system performance. Another important observation is that almost no cheating report was wrongly evaluated as honest even in very hostile settings due to our cautious reputation evaluation mechanism. Therefore, in reality one can observe the results of the trust-distrust propagation step and incrementally adjust the parameters of the system, e.g., increase $F_{special}$, in order to collect enough honest reports for the performance prediction phase. The service selection and ranking can be performed fast and efficiently thanks to the pre-computation of the matching information between QoS of existing services with possible quality requirements of users. Similar to [17], we conclude that *the use of trusted third parties monitoring a relatively small fraction of services can greatly improve the detection of dishonest behavior even in extremely hostile environments*. However, this effectiveness mainly depends on the following properties of the set of service users and their corresponding reports:

1. *The overlaps among the set of users of each service*, i.e., whether the services monitored by trusted agents have many users who are also consumers of other services.
2. *The inconsistency in the behavior of users*, i.e., whether a user is honest while reporting on a service but behaves dishonestly on other cases.

These factors suggest that *we should deploy trusted agents to monitor the QoS of the most important and most widely-used services* in order to get a “*high impact*” effect when estimating behaviors of users. Currently as the user reports are distributed uniformly for services in our experiments, we have not yet taken into account this factor. However, we have employed another technique to fight back the inconsistency behaviors of users by *detecting cheaters first and evaluating honest users later* in the preprocessing phase. This helps us to collect all possible evidences against cheaters by making use of the distrust propagation among reports at the first place. The philosophy behind is that it would be better to filter out a certain number of honest users rather than accept some dishonest reports. However, lots of credible users will be accidentally detected as cheating because of the similarity between their reports with other ones produced by well-disguised liars in the system. Thus, in the honest detecting (trust-propagation) phase, we also give these users a second chance to prove their honesty provided they have produced lots of credible reports which could be certified by trusted reports and other honest users. Additionally, other techniques can also be utilized to make this method more robust. For example, we can *pre-select the important services to monitor and increase the number of them as well as keep the identities of those specially chosen services secret and change them periodically*. Thus, cheaters will not know on which services they should report honestly in order to become high-reputation recommenders and have to pay a very high cost to have great influences in the system. In reality, this also help us to reduce the cost of setting-up and maintaining trusted agents as we only need to deploy them to monitor changing sets of services at certain time periods. Moreover, we can extend our algorithm so that *neighboring registries are allowed to exchange with each other the evaluated credibility of users* to further find out other possibly well-disguised cheaters who frequently change their behaviors. Note that the adjacent registry peers in our system are assigned to manage services with similar functional characteristics and therefore they are likely to attract comparable sets of users in the system [8].

6 Conclusion

In this paper, we have introduced a new QoS-based web service selection and ranking algorithm with trust and reputation management support. We have shown that our selection and ranking solution yields very good results in most cases. As the proposed reputation management mechanism is robust against various cheating behaviors, the results are generally of good quality even in hostile situations in which many different types of cheaters make up a high percentage of the overall users and report values with remarkable variances. By combining a trust-distrust propagation approach with a data-mining method, we could filter out almost all cheaters and find out honest reports to be used in the quality prediction process with very high probability. However, there are a lots of open issues and improvements we can apply to the current model. First of all, the selection of services to be monitored by trusted agents is an

interesting point not yet to be mentioned. Another open problem is how to accurately predict the performance of newly published services with only few QoS reports and how to motivate users to evaluate services' performance and submit their feedback to the service search engine. The selection of an optimal configuration for many design parameters of our proposed solutions is also an important question to be studied in further. Additionally, in each iteration of the trust-distrust propagation, the comparison should be done between an unmarked report with the average of current honest/cheating ones to make the credibility evaluation more accurate since the reports are generally different from each other. Also, after this propagation step, there maybe a number of users whose real behaviors are not revealed due to insufficient evidences while their marked reports include both cheating and honest ones. It is possible to take this factor into account while clustering and evaluating the credibility of various report groups. As part of future work, we will also use QoS properties as ranking criteria for service queries without explicit QoS requirements. Another plan is to develop a so-called meta-behavior model of users, which is more general to describe user behaviors with various possible probabilistic responses and to obtain analytical results of the proposed solution. Last but not least, we are going to deploy our algorithm in a decentralized setting to observe the effectiveness of our trust and reputation techniques where there are many registry peers exchanging among each other the information of users and services' quality data.

References

1. A. Jøsang, R. Ismail and C. Boyd: A Survey of Trust and Reputation Systems for Online Service Provision, *Decision Support Systems*, 2005 (to appear).
2. Z. Despotovic and K. Aberer: Possibilities for Managing Trust in P2P Networks, *EPFL Technical Report No. IC200484*, Switzerland, November, 2004.
3. M. Tian, A. Gramm, T. Naumowicz, H. Ritter, J. Schi: A Concept for QoS Integration in Web Services, *Proceeding of WISEW'03*.
4. S. Ran: A Model for Web Services Discovery with QoS, *ACM SIGecom Exchanges*, Vol. 4, Issue 1 Spring, pp. 1-10, 2003.
5. M. Ouzzani, A. Bouguettaya: Efficient Access to Web Services, *IEEE Internet Computing*, Mar./Apr., pp. 34-44, 2004.
6. C. Patel, K. Supekar, Y. Lee: A QoS Oriented Framework for Adaptive Management of Web Service-based Workflows, *Database and Expert Systems 2003 Conf.*
7. E. M. Maximilien and M. P. Singh: Reputation and Endorsement for Web Services, *SIGEcom Exch.*, 3(1):24-31, *ACM Special Interest Group on e-Commerce*, 2002.
8. L.-H. Vu, M. Hauswirth and K. Aberer: Towards P2P-based Semantic Web Service Discovery with QoS Support, *Proceeding of Workshop on Business Processes and Services (BPS)*, Nancy, France, 2005 (to appear).
9. N. Srinivasan, M. Paolucci, K. Sycara: Adding OWL-S to UDDI, Implementation and Throughput, *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition*, USA, 2004.
10. G. Dobson: Quality of Service in Service-Oriented Architectures, 2004, <http://digs.sourceforge.net/papers/qos.html>.
11. Z. Chen, C. Liang-Tien, B. Silverajan, L. Bu-Sung: UX - An Architecture Providing QoS-Aware and Federated Support for UDDI, *Proceedings of ICWS'03*.

12. A. S. Bilgin and M. P. Singh: A DAML-Based Repository for QoS-Aware Semantic Web Service Selection, *Proceedings of ICWS'04*.
13. S. Kalepu, S. Krishnaswamy and S. W. Loke: Reputation = f(User Ranking, Compliance, Verity), *Proceedings of ICWS'04*.
14. Y. Liu, A. Ngu, and L. Zheng: QoS Computation and Policing in Dynamic Web Service Selection, *Proceedings of WWW 2004 Conf.*
15. F. Emekci, O. D. Sahin, D. Agrawal, A. E. Abbadi: A Peer-to-Peer Framework for Web Service Discovery with Ranking, *Proceedings of ICWS'04*.
16. J. Day and R. Deters: Selecting the Best Web Service, *the 14th Annual IBM Centers for Advanced Studies Conf.*, 2004.
17. R. Guha and R. Kumar: Propagation of Trust and Distrust, *Proceedings of WWW 2004 Conf.*
18. M. Richardson, R. Agrawal, P. Domingos, Trust Management for the Semantic Web, *Proceedings of ISWC'03*, LNCS 2870, p.p. 351-368, 2003.
19. K. Aberer and Z. Despotovic: Managing Trust in a Peer-2-Peer Information System, *Proceedings of ACM CIKM'01*.
20. A. Whitby, A. Jøsang and J. Indulska: Filtering Out Unfair Ratings in Bayesian Reputation Systems, *Icfain Journal of Management Research*, Vol. IV, No. 2, p.p. 48-64, Feb. 2005.
21. F. Cornelli, E. Damiani, S. C. Vimercati, S. Paraboschi and P. Samarati: Choosing Reputable Servents in a P2P Network, *Proceeding of WWW 2002 Conf.*, USA.
22. K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt: P-Grid: a Self-Organizing Structured P2P System. *ACM SIGMOD Record*, 32(3), 2003.
23. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, Semantic Matching of Web Services Capabilities, *Proceedings of ISWC'02*.

An Integrated Alerting Service for Open Digital Libraries: Design and Implementation

Annika Hinze¹, Andrea Schweer², and George Buchanan³

¹ University of Waikato, New Zealand
a.hinze@cs.waikato.ac.nz

² University of Dortmund, Germany
andrea.schweer@uni-dortmund.de

³ University College London, United Kingdom
g.buchanan@ucl.ac.uk

Abstract. Alerting services can provide a valuable support for information seeking in Digital Libraries (DL). Several systems have been proposed. Most of them have serious drawbacks, such as limited expressiveness, limited coverage, and poor support of federated and distributed collections.

In this paper, we present the detailed design and implementation for a comprehensive alerting service for digital libraries. We demonstrate typical user interactions with the system. Our alerting service is open to other event sources, supports a rich variety of event types, and works on distributed as well as on federated DL collections.

1 Introduction

For several years now, there has been increasing demand for a comprehensive alerting services for Digital Libraries [7, 9]. The alerting services supported by proprietary digital libraries, such as Springer Link alert or the ACM digital library TOC alert, are widely known; unfortunately, they have restricted coverage and very limited expressiveness for the subscriptions (e.g., Table-of-Contents (TOC) of a specified journal). These proprietary digital library systems can be contrasted against open digital library systems that provide extendible and flexible tools – supporting varied file formats, custom services and connection between separate libraries.

Recently, existing open digital library systems have started to incorporate alerting functionality. Most of them offer only a restricted focus (e.g., stored search on metadata). Alerting services in Digital Libraries (DL) could inform users about new collections or changes in the classification scheme of a library as well as about new or changed documents. Users express their interest in these *events* through *subscriptions*. The service filters all events and notifies users accordingly. An ‘open’ alerting service should be accessible to a wide variety of digital library systems and extend the flexible principles of open digital libraries briefly introduced above – e.g., by supporting a flexible model of which events may occur, consistently handle distributed and federated libraries and provide

event feeds to differing alerting services. However, to provide a full range of the events that may occur in a specific digital library, the system must also be integrated with the DL – as explained in [5].

This paper introduces the first distributed alerting service that offers sophisticated user subscriptions over a wide range of sources without the limitations of its predecessors: The service supports a wide range of event types using content-based filtering; it is open to external event sources, and notifies users via email, web page or an RSS feed. We present a detailed requirements analysis, our alerting system in use, details of the service design and results of an evaluation. A companion paper [5] describes some of the librarianship difficulties highlighted by the work reported in this paper, omitting the technical content presented here.

The body of this paper commences with the results of our requirements analysis. This section is followed by a demonstration of our Greenstone alerting service in use (Section 3). We subsequently discuss in turn its architecture (Section 4) and detailed design (Section 5). In Section 6, we outline the user-centered and technical requirements that influenced the design of our service. Related work is discussed in Section 7. Our paper concludes with an outlook to future research.

2 Requirements

In the course of designing the Greenstone alerting service, we conducted a number of studies to identify the requirements for the system. One of the key challenges was identifying the different types of events and notifications that a comprehensive service should supply. This part of the design was driven by the results of two studies: a user survey and a claims analysis of the intended design. The latter was based on use cases identified in collaboration with the developers of Greenstone. From these sources, we also developed the user-centered functional requirements for the alerting service. The technical requirements were drawn from an analysis of existing digital libraries. Table 1 summarises the requirements we have identified. We now discuss the requirements in detail, ordered according to their sources.

Table 1. Requirements for the Greenstone alerting service

Functional Requirements	Technical Requirements
F1. find items	T1. provide content-based notifications
F2. add, edit, delete subscriptions	T2. provide customizable notifications
F3. publish event descriptions	T3. support all Greenstone event types
F4. publish event messages	T4. use familiar metaphors for user interface
F5. view notifications	T5. support different kinds of event sources
	T6. event sources on several abstraction levels
	T7. support flexibility of Greenstone set-ups
	T8. support distributed/federated collections

General considerations: A number of general requirements can be drawn from experience with previous alerting services: Users must be able to find items of interest (F1). They must be able to create new and edit or delete existing subscriptions (F2). The providers must be able to publish descriptions of the events they offer (F3), and they must be able to send event messages to the alerting service (F4). Finally, users must be able to view their notifications (F5).

User studies: The requirements that were identified based on the results of the user studies are: A major concern was that the service could notify users about too many irrelevant events (false positives). It is especially remarkable because no option in the questionnaire corresponded to this problem. The users concerns are taken into account as the requirements that subscriptions should be as fine-grained and as similar to conventional Greenstone usage as possible. In addition to that, notifications have to be as unobtrusive as possible. Both goals can be reached by providing customizable, content-based notifications (T1 & T2).

DL Scholarship: Based on our experiences with digital libraries, we believe that the alerting service should have the following characteristics to be fully integrated into a digital library: It should provide notifications about a wide selection of events occurring in the digital library (T2 & T3). It should stay consistent with the users conceptual model of the digital library (i.e., creating a subscription and receiving a notification should be similar to using the other services of the digital library) (T4). It should integrate with the infrastructure of the digital library (i.e., seamlessly support distribution and federation of the DL) (T8).

Greenstone specific: A number of requirements were developed to address the special features of the Greenstone digital library system: Greenstone allows for the combination of different internal and external sources for a DL collection. The alerting service should be similarly open and support other event sources in addition to Greenstone DLs (T5). Greenstone is very flexible as to which document formats can be stored and retrieved, the metadata formats that can be used, the collection structure and the service configuration. The alerting service should therefore place as few constraints as possible on the configuration of the collections it can be used for (T6 & T7).

The requirement T8 creates particular challenges for the alerting service in the Greenstone context: To support Greenstone's distributed nature, the alerting service itself has to be distributed to a much higher degree than present in current alerting services in the context of digital libraries. Ideally, users should be able to create one single profile and then transparently add subscriptions for different Greenstone installations to it. In addition to that, it should be possible to subscribe to events from different collections or hosts using one single subscription (for example, notify me about all new collections).

To address requirements T2 and T3, we analyzed Greenstone 3 to identify useful event types. In the context of DLs, events refer to state changes in all objects in the DL software: such as collections, documents, and the software itself. We identify all objects in this context and list the creation and destruction of each kind of object (where applicable), as well as all ways these objects can change. Table 2 lists all 24 types of events we identified for Greenstone 3.

Table 2. Event Types Identified for Greenstone 3

Event Type	Details
software	new release, new bug, bug resolved, new patch
host	new host, host deleted
interface	new interface, interface deleted
site	new site, site deleted
collection	new collection, collection deleted, collection rebuilt
document	new document, document deleted, content of document changed, metadata of document changed
part of document	new part of document, part of document deleted, content of part of document changed, metadata of part of document changed
service	new service, service deleted, service-specific event

3 The Alerting Service in Use

In this section, we demonstrate the user side in two typical interactions with the alerting service: first, the initial creation of a subscription (i.e. registering an interest) and, second, the receipt of notifications that match the subscription. For clarity, we will primarily focus on the familiar events of new or changed documents; our full range of events will be introduced in Section 5.

3.1 User Side: Creating Subscriptions

Simple subscriptions: Our user is browsing a digital library collection that frequently has changes and updates made to its content (this often occurs with, e.g., WHO collections). The user has found a document that is of interest to them and that may contain additional information in the future (see Figure 1, left). In order to monitor the evolution of this document, they click on the “Watch this”



Fig. 1. “Watch this” buttons to create subscriptions shown on an individual document page (left) and a classifier display (right)

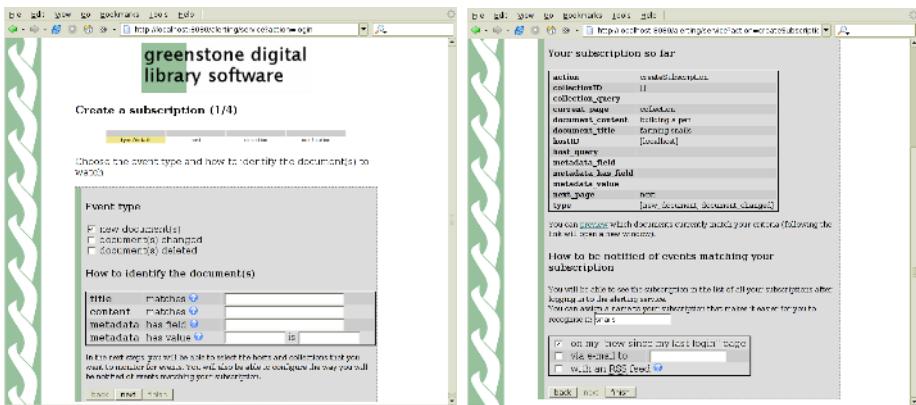


Fig. 2. A user's subscription displayed for editing; Steps 1 (left) and 4 (right) of four

"This" button circled on the left-hand side of the interface in the figure. This simple gesture registers a subscription for the user that will now send them a notification when a change is made to the document.

To find new documents, our user may turn to classifications in Greenstone: They might find a classification that is interesting and relevant to them (see Figure 1, right). Again, they click on the "Watch This" button, and they will be subsequently notified when a change is made to the content of this classification, e.g., when a new document appears in it, a new sub-classification is added or a member document is removed.

In both these cases, the alerting service simply uses this user's default preference for how to notify about the updated information. Simple subscriptions are set up in a browsing interaction style. Obviously, this kind of subscription only works for existing classifications or documents.

Complex subscriptions: Users can also create subscriptions about documents where no classification exists yet, or define sophisticated subscriptions. Here, we use a technique similar to a search query. A subscription is created in four steps. In Figure 2, we can see the first and the fourth step in the process. Firstly, users define the event type and a query that helps identify the involved documents (Step 1, see Figure 2, left). Next, the involved Greenstone hosts (Step 2) and collections (Step 3) have to be defined (by selecting from a list or specifying part of the name). Finally, the means of notification are defined as shown in the screenshot in Figure 2, right. After defining a subscription, notifications about the events may be received by the user; this is described in the following subsection. Users can always edit or delete their subscriptions with immediate effect.

3.2 User Side: Receiving Notifications

Greenstone (GS), being a full-text digital library system, works with periodic explicit rebuilds (re-indexation) initiated by the collection administrator. Thus,

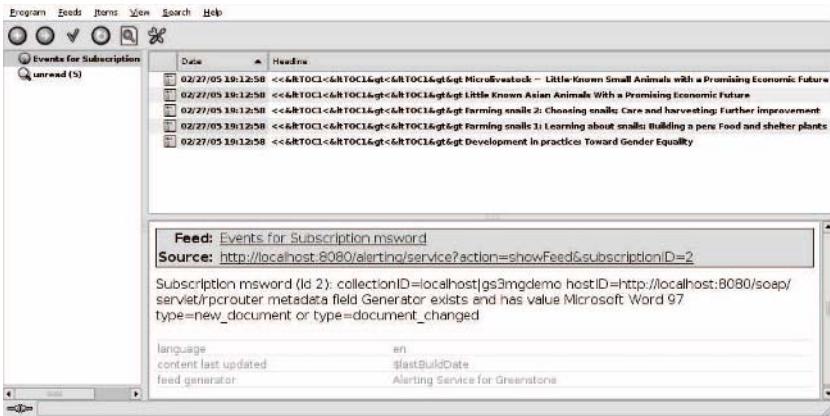


Fig. 3. A Greenstone RSS feed shown in a feed reader

the typical pattern of updates to the library are occasional and large-scale rebuilds, rather than frequent changes to individual documents where updates are performed immediately on the receipt of a changed document. Our Greenstone alerting service is triggered by each rebuild; changes might have occurred in the rebuild that subscribed users may be interested in. Any user who has at least one subscription involving the rebuilt collection may be due to receive notifications about changes to the library.

As discussed before, users can configure the delivery method for notifications in their subscription configuration. At present, we support email and RSS feed notification for personal delivery. An example RSS feed is shown in Figure 3, accessed using the Liferea feed reader (<http://liferea.sourceforge.net>).

Alerts may also be displayed in the Greenstone library interface in a general or personalized way. The general interface may provide a ‘new accession page’ and ‘new/changed’ highlight for particular documents. The personal page may display a user’s recent notifications after the user has logged in to Greenstone.

4 Architecture of the Alerting Service

We now discuss the server side of the alerting service, providing an overview of our alerting architecture and its distribution. Detailed descriptions of the components and their design will be given in the following section. We chose the latest generation of the Greenstone DL software as the basis for building our system. Greenstone is a well-established software system for delivering digital library services; the alerting service builds upon Greenstone’s modular architecture [2].

General Architecture: The general architecture for the alerting service is shown in Figure 4. Existing DL components in Greenstone are identified in white, whilst the new elements are highlighted in gray. The alerting sequence [8] comprises

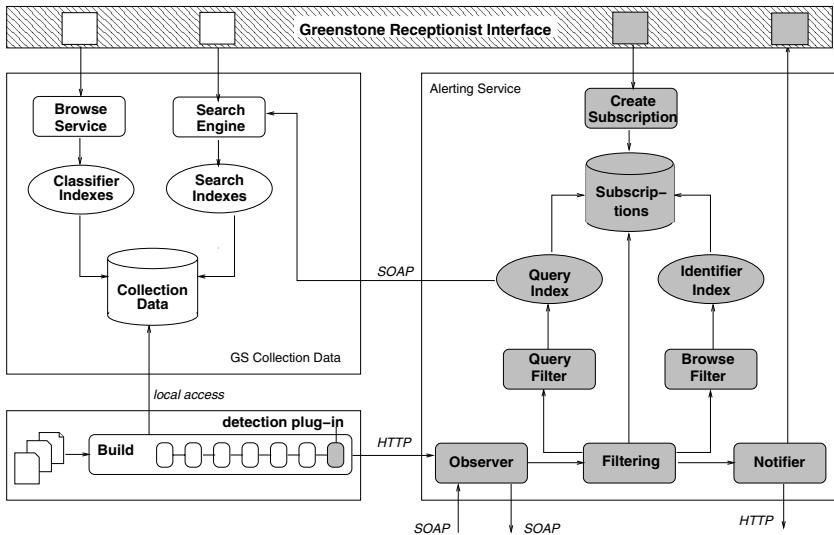


Fig. 4. General architecture of a local Greenstone installation (left) with build process (bottom left) and alerting service (right)

four steps that can be seen at the bottom of the figure: (1) rebuild and trigger of alerting service, (2) observation of event messages, (3) filtering of event messages according to user subscriptions, and (4) notification.

Note that the *Observer* component in itself is distributed: It consists of a generic observer (within the alerting service in Figure 4) and an event detection plug-in for each event type (shown in Figure 4 as last phase in the built process). The alerting sequence is initiated when a collection in the library is (re-)built and changes in the library's indexes and content are identified. Each event is reported to the *Observer* process, which prepares them for processing. The *Filtering* phase then takes each event in turn, and matches it against the filter's own index of subscriptions. If a match is found between a subscription and an event, a notification should be sent to the user who owns the subscription. A match is sent to the *Notifier*, which creates a notification about the event message and sends it to the user according to the user's preferences. The heart of this apparently simple sequence is the *Filtering* phase, which will be discussed in detail in Section 5.

Distributed Architecture: The Greenstone alerting service can be distributed. A distributed architecture for the alerting service is required in three cases: (a) a number of different digital libraries use a common alerting service; (b) a library's content is distributed; or (c) the alerting service is on a separate computer to the DL server. Greenstone itself is designed to be distributed if required [1]. Therefore, Case (c) is simple since any connection in our architecture (see Figure 4) may be between processes on separate machines. Beyond this basic separation,

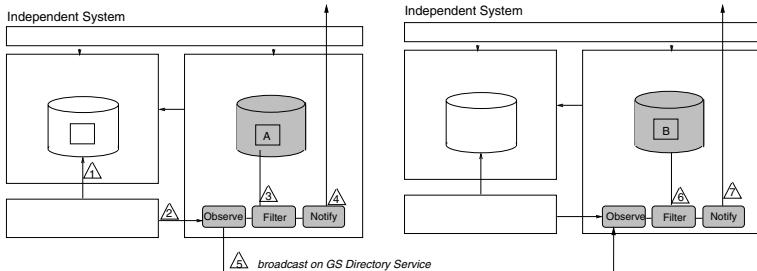


Fig. 5. Federated collections on distributed Greenstone installations. Two independent systems shown (simplified).

we make a distinction between *federated* collections, Case (a), and *distributed* collections, Case (b). For the forwarding of event messages to distributed or federated Greenstone servers, we have created the *Greenstone Directory Service (GDS)*. The GDS provides a common message-forwarding-infrastructure for all alerting distribution scenarios. Here, we give a brief resume of GDS – for complete details see [4]. The GDS is constructed as an independent network of Directory nodes that together form a tree structure. Each tree of GDS nodes provides a separate community, which can send and receive messages to and from any digital library registered with a node of the GDS tree. GDS messages are transmitted in an XML format across TCP/IP connections. Each node receives messages from its children, and distributes received messages forwards to all its other children, and upwards through the tree for propagation across the whole GDS network of which the node is a member. Messages are transmitted through the GDS network when a collection is rebuilt – being sent into the GDS server with which the host digital library server is registered. Note that a GDS transmission node does not itself match or filter events against subscriptions – it purely acts as a transmitter in a communication network. The new build process we implemented in Greenstone supports notification of changes made during a collection rebuild through the GDS service, and we have also implemented the receipt and processing of remote collection changes against locally stored subscriptions in a discrete subscription filtering client application. Other digital library systems can readily be extended to support the transmission of change messages into the GDS network – it is not specific to the Greenstone software. Support for EPrints and D-Space – both popular digital library systems which we will meet later in this paper – is anticipated in the near future.

The case of federated collections is shown in Figure 5. Two Greenstone hosts are shown with their Greenstone software and alerting services. The left-hand Greenstone server hosts a collection that the two subscriptions A (left) and B (right) are interested in. When a change occurs in the collection (by rebuilding the collection – see Step Δ), the last phase of the build process (i.e., the detection plug-in) triggers the observer (Step Δ), which, in turn, forwards the event message to the local filter component and broadcasts it also on the Green-

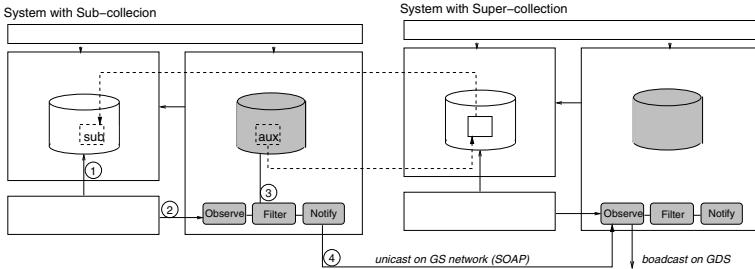


Fig. 6. Distributed collections on distributed Greenstone installations. System with sub-collection (left) and system with super-collection (right).

stone Directory Service (Step \triangle). Locally, the event is filtered according to subscription A (Step \triangle) and the respective user is notified (Step \triangle). For the distributed case, every alerting process on a Greenstone host connected to the Greenstone Directory Service receives the broadcast event message. We now follow the alerting process through the system shown the right hand side of Figure 5: The observer receives the event message and forwards it to the filter. The event is filtered according to subscription B (Step \triangle) and its user is notified (Step \triangle). We can see that the subscriptions are kept locally at the alerting server that initially receives them.

The case of distributed sub-collections on a separate server is handled differently (see Figure 6): The super-collection (right) holds a reference to its sub-collection (left), indicated by a dashed line between the GS collection databases. Sub-collections have no knowledge of their super-collections. Consequently, they cannot send out messages referring to the changed collection (the sub-collection might have a different local name); what is needed is an event message that refers to the super-collection. The alerting system creates an auxiliary subscription for the sub-collection (left), which refers to the super-collection, represented by a dashed line. On rebuild of the sub-collection (Step ①), the local observer is triggered (Step ②). The matching auxiliary subscription is identified (Step ③) and a notification is sent to the super-collection's server (Step ④) via the SOAP-based Greenstone network. The Greenstone network only connects servers with sub/super-collections. The observer on the super-collection side receives the message and identifies the respective super-collection. It then sends out an event message for the super-collection over the GS Directory Service. The further process is identical to the event handling in the distributed case as described before (starting at Step \triangle). For sub/super-collections, subscriptions are forwarded, in opposition to event forwarding in the distributed case.

5 Design Details

This section describes in detail the components of the Greenstone alerting service introduced in the previous section. The design of our alerting service supports 30

different event types, for instance: document events, metadata events, collection events, index events, category events, classifier events, system events, and software events. The event types observe different actions (such as new, changed, deleted items) supporting different ways of identifying the affected items. For conceptual clarity of description, here we focus on new and changed documents. For comprehensive coverage of the service's design, refer to [10].

5.1 Design: Creating Subscriptions

The Greenstone alerting service provides a web interface to create subscriptions. All subscriptions are conjunctions of predicates. Predicates have a type, a field name, and a value. There are four different kinds of predicates: Equality predicates, metadata predicates, substring predicates, and query predicates. Each predicate refers to a single field name. If more than one predicate in a subscription refers to the same field, these are evaluated as disjunctions.

Equality predicates are currently used for identification of hosts, collections, and documents. Substring predicates are currently only used for hosts and collections. Metadata predicates refer to metadata fields and values. Metadata predicates are satisfied if the document has the specified metadata field and the specified value is equal to that of the metadata field. The field name may be freely defined by the user. Query predicates can be defined for title and full-text of documents; they have free-defined Greenstone queries as values.

5.2 Design: Observing Events

Each collection in Greenstone consists of the documents' full-texts and accompanying metadata. During a rebuild, this information is processed into the standard METS format and stored in a relational database. Classification and content indexes are also generated for the collection. Each collection and each document has three time-stamps. For alerting, the following time-stamps are relevant: `accession` date and last (re)build for a collection, and time-stamps for `accession`, `modification` and (re)`indexed` for documents.

When a collection is rebuilt, the collection, new, and changed documents receive the same (re)`built`/(re)`indexed` time-stamp. New documents are identified by their `accession` time-stamp being identical to their (re)`indexed` time-stamp. To detect changed documents, the `modification` and (re)`indexed` time-stamps are compared. On each rebuild, an alerting sequence is initiated. Greenstone 3 uses incremental rebuilds; the detection plug-in at the end of the build process detects the events of changed or new documents. It then creates event messages containing attribute-value-pairs. The event message holds information about the type of event, the affected document's ID, and the collection ID. The message is sent via HTTP to the observer component of the alerting service.

The generic observer component provides an interface for internal and external event messages: messages from local builds are received through HTTP, whilst events from builds on other hosts are received through SOAP. The event messages are passed on to the filtering component of the alerting service.

5.3 Design: Filtering Event Messages

The filter component tests incoming event messages against the subscriptions stored in the database. We use a hybrid implementation of the equality-preferred-algorithm, which is in turn an extension of the counting algorithm [6]: The counting algorithm tests each predicate in a set of subscriptions only once, counting the number of successful predicate matches for each subscription. A given subscription is matched by the incoming event message if its number of matched predicates equals the total number of its predicates. The equality-preferred-algorithm tests all equality predicates first and then proceeds to further predicates for the subscriptions using the counting algorithm.

We use a hybrid equality-preferred-algorithm in three phases. The filtering component holds a special index of subscriptions, clustering them according to their equality predicates, i.e., all subscriptions with equality predicates regarding the same set of field names are clustered together (see identifier index in Figure 4). In Phase 1, the corresponding clusters for the incoming event are identified. Corresponding clusters are those which refer to fields that are contained in the event message. Then, the equality predicates in these clusters of subscriptions are evaluated (using the clusters' hash indexes on the values). The result is a list of partially matched subscriptions.

In Phases 2 and 3, all other predicates are tested using the counting algorithm. In Phase 2, all non-equality predicates are filtered and a counter is maintained for each affected subscription. Non-equality predicates are metadata predicates, substring predicates, and query predicates. For query predicates, we use collection-inherent DL searches for the stored queries (see query index with access to GS search engine in Figure 4, left). Similarly, for metadata predicates we use Greenstone's metadata retrieve service. The Greenstone alerting service determines the appropriate query service offered by a collection by applying heuristics for typical queries on a field, e.g., a predicate regarding the title field uses a title search in a collection. If no appropriate service is found, the predicate is not satisfied.

In Phase 3, the number of matching non-equality predicates are compared to the total number of non-equality predicates in each partially matched subscription. In addition, all subscriptions without equality predicates are considered. Information about matching subscriptions is written to the database to be used for notifications.

Our design constitutes a novel approach to combine filter and search functionalities; this is necessary because possible search methodologies for future collections may not be known at the time when the subscription is defined.

5.4 Design: Creating Notifications

Notifications can be delivered to the user in a general or personalized way. General delivery uses, e.g., a ‘new accessions’ page in the Greenstone interface, and ‘new’ highlights for collections or documents in the interface. We maintain a Greenstone 3 alerting module for presentation of general notifications; the GS3

user interface is created from such modules dynamically at runtime [2]. Each interface module (including the alerting module) communicates with the core DL interface code through SOAP. These general notifications require that recent notifications are recorded in Greenstone's relational database for future use.

Alternatively, the users may receive notifications through a message (email) or RSS feed. Implementation of these notification is straightforward, and require no further explanation. An example has been shown in Figure 3.

6 Evaluation

In this section, we present the results of our evaluation of the alerting service. This section consists of two principal parts: first, the discussion of the compliance of our design with the original requirements introduced earlier in this paper; second, an analysis of the performance of the system in use.

6.1 Evaluation: Design

The requirements for our alerting service were introduced in Section 2. We will first discuss the functional requirements F1 to F5: The service supports the finding of items through a simple addition to the standard DL build process [5]. We have demonstrated the editing of subscriptions (F2), publication and viewing of events (F3–F5) in Section 3.

Research in event-based systems has indicated that users frequently have problems defining effective subscriptions to represent their interests (reflected in requirements F2 & T4, see Table 1). To overcome the syntactic problem of defining even basic queries, we introduced simple interface features such as the “Watch This” button. For sophisticated subscriptions, we provide an advanced query subscription interface. The learning demand on the user is reduced by providing an analog of Greenstone's interactions, mirroring browsing and querying. Further user testing needs to be conducted to refine this approach.

The technical requirements T1 to T8 are also addressed by our design. We use the digital library itself to determine content-based events (T1), and as seen in Section 3 customizable notifications (T2). Our close relationship to the existing DL system readily supports a user interface similar to the familiar DL controls (T4) and a range of events consistent with the capacities of the underlying DL systems (T3). The alerting service's open format for events readily provides open access to a variety of event sources (T5) and levels of abstraction (T6). Our use of subscription forwarding (in the case of distributed libraries) and event forwarding (for federated libraries) ensures consistency where such collections exist (T8) and a variety of DL configurations (T7) – in the latter case supported further by our open event format.

6.2 Evaluation: Performance

We wished to study the performance of the distributed alerting service in practice. There were three separate areas that we wished to evaluate in terms of

performance: event detection, filtering of incoming events against locally stored subscriptions, and finally the distribution of events across the Greenstone Directory Service network.

Before reporting our results in detail, it is worth clarifying the rate of change that may be expected in a digital library. The Humanity Development Library (or HDL) is produced by the United Nations and distributed on CD-ROM. The current public distribution contains over 160,000 separate pages, each indexed as a separate document. Of this material, some 50% has been updated or added since the first version seven years ago. In other words, the number of items changed per day is somewhat under 50. This rate of change is typical for the various United Nations collections that are supported by Greenstone, and the HDL is of median size. These UN collections have a higher change rate for existing documents than is typical for most digital libraries. It follows from this sort of volume of change that the challenge for the distributed alerting service comes less from the frequency of changes in each library collection, but rather from the potentially vast distribution/federation, the number of libraries involved, and the number of profiles.

Event Detection: For the first tests of the alerting service, we wished to verify our expectations of the time costs for detecting changes in the digital library – an area where existing data gave little indication of what order of performance one ought to expect.

Detecting a change in a document, or in a classification requires a series of simple database lookups that we expected to be in the order of $O(p)$, where p is the number of document properties (e.g., metadata such as document title – see Table 2). Running a sample build over a collection comprised of a mixture of large plain text, HTML and XML documents (of book size) resulted in an underlying build time of c. 15 seconds per document (mean=15.2s) of which c. 0.2 seconds (mean=0.16s) was spent identifying which, if any, change messages should be forwarded for this document. Clearly, event detection adds only a small cost to the indexation of a document in the library. We believe that this performance can be slightly improved with further optimisation of the build code.

Local filtering: Secondly, the issue of profile matching required study – identifying the process load of processing incoming events against the user profiles stored on a local Greenstone server.

The performance of the counting and equality-preferred algorithms is well known. However, in our implementation we extended the original algorithm, which compared only numbers. We had to support full-text search and string comparison to satisfy the full range of events for Greenstone and other digital libraries. Neither of these forms of comparison have any relationship with the number of subscriptions to be matched, so we should still anticipate overall performance characteristics of the local filtering to be similar to the original algorithms [6]. The primary expectation from the earlier implementation should be for performance to be in the region of $O(s)$ where s is the number of subscriptions (or, more precisely, unique subscription predicates). Fabret et al's algorithm evaluates any predicate (e.g., “title includes Shakespeare”) only once, even if

Table 3. Results of local filtering tests for Greenstone Alerting Service

Unique predicates (subscriptions)	Time Taken (in sec)
1000 (256)	0.142
2500 (640)	0.182
5000 (1280)	0.243
7500 (1920)	0.338
10000 (2560)	0.437
25000 (6400)	0.783
50000 (12800)	1.465
75000 (19200)	1.981
100000 (25600)	2.479

the predicate is included in a number of different subscriptions. In our tests, we followed their model of considering the performance of their algorithm on the number of unique predicates, rather than (more complex) subscriptions.

We conducted a small experiment to evaluate the local filtering performance – varying the number of predicates in the profile database from 1,000 to 100,000 unique predicates. Predicates were of equal portions of the different available types, created using simulated subscription data. A digital library subscription will typically include three or more predicates (in our simulated test data, an average of four) – e.g., a subscription may list an event type, collection, author name and subject field. At a worst case, 100,000 predicates would typically represent 25,000 subscriptions, if every predicate of every subscription were unique. Real life data would certainly include some predicates that appeared in more than one subscription, increasing the number of subscriptions that this would represent.

The time taken to match all predicates against a single event varied linearly from 0.14s (with 1000 predicates) to 0.44s (10,000 predicates). Approximately 0.10s was fixed overhead. These findings are consistent with those of Fabret et al. However, this test does not represent the whole picture – full-text retrieval predicates are much more costly to evaluate than simple string matching, and we wish to undertake further study to distinguish the costs of different styles of predicate. Unfortunately, we have little data at present to identify the relative frequency of particular types of predicate (tests of document metadata, full-text, etc.) under actual use.

However, the results above demonstrate that the matching of events against user subscriptions is scaleable and consistent with known state-of-the-art algorithms.

Event Distribution: Given the tree structure of the Greenstone Directory Service, one critical limitation could be the throughput capacity of the single node at the root of the tree, through which every event would have to pass. Therefore, our first point of concern was the nominal capacity in practice for a single node within the GDS to receive and forward messages. Using event forwarding over the GDS network (as described in Section 4), each change in a collection is

received only once by a network node, and forwarded once to each immediate child node, and once to its parent node (excepting the case of the root node). Note that GDS nodes do not filter events, but transmit them onwards through the network without further processing. Thus, the primary limitation is in fact the rate of *output* to other nodes.

For testing purposes, the GDS network was run on a small cluster of servers at University College London, consisting of two Apple Mac OS-X computers, two Linux servers and two machines running Windows XP. Three computers ran as GDS servers only, three as Greenstone servers producing Greenstone Alerting messages. Two GDS servers were registered with the third, which acted as the root server. The GDS root server machine was an Apple Mac-OS X G5 computer with 1Gb of main memory. The computers were connected locally through a 10Mbit network, with two (Linux) machines running at a remote site at Middlesex University (UK).

As shown in [3], even such a small network topography can be used to successfully test distributed alerting services, as it has the advantage of a real-world test over a simulation.

We expected the time cost of the distribution of events to be small, but limited by the available network bandwidth and the overhead of establishing connections between GDS nodes. We achieved average point-of-origin to remote (off site) recipient transmission times of 2.3 seconds per event, but further testing indicates that the current implementation could be significantly improved – e.g., at present each message is transmitted as a new, separate connection, and this is clearly wasteful.

7 Related Work

In this section, we review previous work for alerting in digital libraries in proprietary systems, in open DL systems, and in mediator approaches.

Alerting in Proprietary DL Services: Individual alerting services are offered by publishing houses, such as Springer Link Alert (via <http://springerlink.metapress.com>), ACM Table-of-Contents Alerts (via <http://portal.acm.org>), and Elsevier Contents Direct (<http://www.contentsdirect.elsevier.com>). These are solitary, centralized services that neither cooperate with other services nor openly support independent digital libraries. These services provide simple email notifications about new volumes published by the company, rudimentarily tailored to the user's interests. Only coarse-grained selectivity is offered, which may result in readers obtaining a high proportion of notifications of low relevance. Advanced subscriptions, e.g., regarding library organization or classification of documents are not supported. Unlike our Greenstone alerting service, the systems are not open to additional event sources.

Alerting in Generic DL Systems: So far, only two of the popular generic DL systems provide alerting features – D-Space and EPrints. D-Space (<http://www.dspace.org>) is being developed as a reference model for document management systems, and supports the storage and retrieval of electronic documents.

Readers using a D-Space server can place a subscription on a specific collection, which then waits for any documents to be added to the collection. No additional constraints can be added to a subscription – the subscriber is simply emailed a notification each day listing all new documents added to the collection. This can be compared to a simple ‘watch-this collection’ subscription without further filtering in our alerting service.

EPrints (<http://www.eprints.org>) is a simple open source system for providing an internet-accessible document repository. EPrints supports simple subscriptions that alert a reader when a matching document is inserted into the EPrints repository; matches are made against the metadata fields of a document. In the Greenstone alerting service, this can be compared to a basic metadata subscription using a query that is restricted to new documents.

The engineering of the incorporation of the alerting services in EPrints and D-Space has been reported only briefly in the available literature. In contrast to the GS alerting service, their subscription systems are deeply embedded in the DL implementation.

Mediating Alerting Services: Only a few systems have been developed to support open heterogeneous document collections for publish/subscribe features. Here we focus on the two systems that target at DLs: Hermes and Dias.

Hermes [7] is an integrative system that covers heterogeneous services and event sources. Subscription definition focuses on typical queries regarding scientific publications, such as authors, title, or keywords. The service operates independently of any library implementation, using (active) email or (passive) web pages for information access. Typically, such an alerting service would be operated by a scientific library (secondary provider) as a service for its users, notifying about documents provided by primary providers. Unlike our Greenstone alerting service, Hermes only aggregates notifications from different sources and is limited by the underlying types of alerts that it receives. It was this restriction that motivated the work presented in this paper.

Dias [9] adopts the basic ideas of Hermes. Dias is a distributed system based on peer-to-peer communication. The data model of Dias is based on simple free-text documents. Dias’s subscriptions support Boolean queries with proximity operators. We see this text-focussed approach as too limited for an open digital library supporting collections of arbitrary document types (e.g., music, pictures, text documents).

Tools such as Hermes and Dias suffer significantly from not being integrated with the digital library. For example, observation of events and access to documents is problematic. In addition, identifying different versions of the same work is particularly difficult without explicit knowledge of the underlying structure of collections or a set of valid document identifiers. Similarly, approaches that rely on poor sources such as the active support from publishers and DLs or on monitoring publishers’ web-pages to extract event information cannot hope to lead to sophisticated event notifications.

Summarizing the related work, most existing systems only support the detection of new documents. EPrints additionally supports changed documents.

Advanced subscriptions, e.g., regarding indexes or classifications, are not available. Often systems are implemented in a centralized manner. Support for federated or distributed collections could not be found. Only mediating systems support open event sources; unfortunately they receive only poor support and are extremely limited in their access to pertinent and detailed data.

8 Conclusions

This paper proposed the design and implementation of a comprehensive alerting service for digital libraries. We have shown that existing alerting services for DLs have considerable shortcomings: limited types of supported events; limited range of subscription model and notification options; no support for distributed collections; restricted support for federated collections.

To address these limitations, we presented the detailed design of the Greenstone alerting service, describing both stand-alone and networked implementations. The Greenstone alerting service supports a much wider range of event types than previous systems, and supports events in federated and distributed collections. Our alerting service is open to other event sources and providers.

Our design for a comprehensive alerting service can readily be applied to a wide range of DL systems, as it reuses existing DL components, and requires only protocol-level access to the DL to which it is attached. In addition, by using existing DL system functionality, it minimizes both programming and run-time demands of the alerting service. Using well understood principles from event-based systems, efficiency can be achieved without building extensive specific indexes for alerting. We demonstrated the scalability of our approach through a series of test of both the distributed event forwarding system and the subscription filtering component.

We plan to evaluate further performance optimization strategies of the filter algorithm. We also plan to further analyse the scalability of the distributed service under a high load of subscribers, particularly when matching a high proportion of content-text subscriptions. Initial results are promising, but we wish to scrutinise this particular issue further to ensure scalability. Another extension involves a modification of the Greenstone protocol: We plan to develop a structured methodology to determine the appropriate query service to be used for evaluating a given filter predicate.

Acknowledgements. This work was supported by the University of Waikato and EPSRC grant (GR/S84798).

References

1. D. Bainbridge, G. Buchanan, J. R. McPherson, S. Jones, A. Mahoui, and I. H. Witten. Greenstone: A platform for distributed digital library applications. In *Proceedings of the ECDL*, Sept. 2001.
2. D. Bainbridge, K. J. Don, G. R. Buchanan, I. H. Witten, S. Jones, M. Jones, and M. I. Barr. Dynamic digital library construction and configuration. In *Proceedings of the ECDL*, Sept. 2004.

3. S. Bittner and A. Hinze. Classification and analysis of distributed event filtering algorithms. In *Proceedings of COOPIS*, October 2004.
4. G. Buchanan and A. Hinze. A distributed directory service for Greenstone. Technical Report 1/2005, Department of Computer Science, University of Waikato, New Zealand, Jan. 2005.
5. G. Buchanan and A. Hinze. A generic alerting service for digital libraries. In *Proceedings of the JCDL*, June 2005.
6. F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Efficient matching for content-based publish/subscribe systems. Technical report, INRIA, France, 2000.
7. D. Faensen, L. Faulstich, H. Schweppe, A. Hinze, and A. Steidinger. Hermes – a notification service for digital libraries. In *Proceedings of the JCDL*, June 2001.
8. A. Hinze and D. Faensen. A Unified Model of Internet Scale Alerting Services. In *Proceedings of the ICSC (Internet Applications.)*, Dec. 1999.
9. M. Koubarakis, T. Koutris, C. Tryfonopoulos, and P. Raftopoulou. Information alert in distributed digital libraries: The models, languages, and architecture of Dias. In *Proceedings of the ECDL*, Sept. 2002.
10. A. Schweer. Alerting in Greenstone 3. Master's thesis, University of Dortmund, Germany, May 2005.

Workflow Data Guards

Johann Eder and Marek Lehmann

University of Klagenfurt,
Dep. of Informatics-Systems
`{eder, marek}@isys.uni-klu.ac.at`

Abstract. Workflow management systems (WfMSs) frequently use data to coordinate the execution of workflow instances. A WfMS evaluates conditions defined on data to make the control flow decisions i.e. selecting the next activity or deciding on an actor. However, data - within and outside of a running workflow instance - may change dynamically. Modifications of data needed for past control flow decisions may invalidate these decisions. We analyze the desired synchronization policies, and propose a mechanism called data guard to selectively guarantee that significant changes in data are recognized and handled by the data management system to ensure correctness of workflow execution in face of asynchronous updates.

1 Introduction

Workflow systems frequently integrate autonomous information systems, using and manipulating data from various heterogenous sources. These data are crucial for the correctness of a workflow execution, because control decisions depend on them. These data can be dynamically modified within and outside of an active workflow instance. Unfortunately, this is frequently ignored and replaced by an assumption that only workflows are accessing data and that the data modified by one workflow are not accessed by another one [1].

The modifications of data can have different impacts on a running workflow instance. We can classify the effects of changes as follow:

- The changes have no effect (e.g. modifications concern data already used in the workflow and not needed anymore)
- The running instance uses the new values (e.g. contact a customer using his or her new phone number).
- The running instance needs the old values (e.g. when the cancellation policy at the time of contracting is relevant, not the latest version.)
- The change invalidates parts of the workflow and leads to an exception handling (e.g. a control flow decision based on a data value made during the workflow execution can be invalidated if this data value changes after making the decision.)

To illustrate the last case we use a simplified car insurance claim handling workflow definition as modelled in Fig. 1. A possible instance of this workflow is

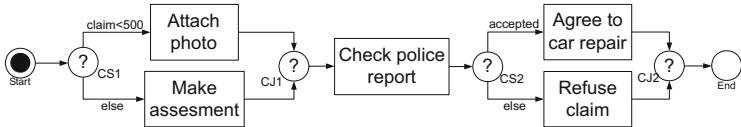


Fig. 1. Simplified example of a car insurance claim handling workflow

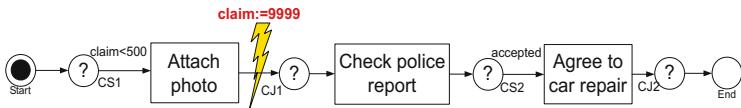


Fig. 2. Sample instance of the workflow definition in Fig. 1

presented in Fig. 2. A control flow decision was made in the cond-split node CS1 based on a claim value less than 500. All the following activities were performed with such an assumption. If the claim value has changed (perhaps outside of a running workflow) after the node CS1, then the agreement to the car repair is possibly invalid, and thus the whole workflow is invalid.

In most cases the existing WfMSs do not offer simple means for checking such errors. The workflow designers usually have to find some workarounds and manually program all the checks. The possible methods may include:

- Guarantee, in all activities within a workflow, that data values will not be changed in a way that may invalidate the workflow. This method does not prevent data changes made externally to the workflow.
- Implement condition checks in activities: makes activities process specific, reduces reusability and makes maintenance more difficult.
- Pre- and postconditions: frequently it is possible to define pre- and postconditions to activities, but not to their particular occurrences (steps) in the workflow definition. This again reduces the reusability of activities.
- Control flow approach: a workflow designer can use conditional nodes to recheck the data in the workflow definition before the critical activities and take an appropriate action if an error has been detected. This approach can lead to the explosion of the workflow complexity and does not solve the problem, because data can change at any time outside of a workflow.
- Lock the data used to make the control flow decision for the execution time of the rest of the workflow instance. But workflow instances can take very long (e.g. months) and locks held for such a long time are unacceptable.

The methods outlined above do not solve the general problem and introduce many new ones. From the analysis above, it is also clear that there is not one simple solution to care for all cases. We propose workflow data guards as a technique for monitoring changes and for giving the possibility to react to changes in the right way. The rationale for our mechanism is that in a dynamic workflow

environment we cannot prevent all data changes, but we would like to be able to detect and to react to some of them in selected parts of a workflow.

The work we present here is a continuation of our work on data aspects in workflow systems. In [5] we proposed a uniform treatment of all kinds of business data in workflows and offered the transparency of data location and logical and physical data independence of workflow systems by using specialized wrappers around external data sources called data access plug-ins. In [6] we presented a policy based mechanism for synchronizing copies of external data made in the workflow repository.

The remainder of this paper is as follows: Sec. 2 describes the concept of workflow data guards. In Sec. 3 we discuss different methods of activating and deactivating the guards. We show how to check conditions of active guards in Sec. 4, discuss related work in Sec. 5 and finally draw some conclusions in Sec. 6.

2 Workflow Data Guards

We represent both workflow definitions and workflow instances as full-blocked workflow graphs and explore the concept of data guards in detail for this workflow model. The concept of workflow guards and the principal definitions, nevertheless, should be applicable to most workflow models we are aware of.

Full-blocked workflow graphs are directed acyclic graphs with two kinds of nodes: activities and control nodes. The edges determine the execution sequence of nodes. Control nodes describe basic workflow control structures: conditional (cond-split and cond-join nodes symbolized by a circle with a question mark) and parallel execution (par-split and par-join nodes symbolized by a circle with two parallel lines). In a full-blocked workflow graph, with each split node is associated exactly one join node of the same type and each join node has exactly one corresponding split node of the same type. A workflow instance type graph is a subgraph of a workflow definition graph, where each cond-split node has only one adjacent successor.

A *workflow data guard* is a constraint defined over specified parts of a workflow. It is specified in a workflow definition and consists of a condition, a set of activities which activate the guard for the first time (activation set) and a set of activities guarded (guarded set). Its condition is a boolean expression defined on data used within this workflow.

A workflow data guard instance can be in one of three states at runtime: inactive, active and deactivated. Inactive is the initial state when a workflow instance is started. The guard instance stays in this state until an activity from its activation set is started. Later it can be deactivated and activated again, etc. according to the activation policy detailed below. Only when the data guard instance is active its condition is checked and a guard violation can be detected.

The activation policy of data guards is as follows:

- Guards are inactive, when a workflow instance starts.
- Launching activities of the activation set make an inactive guard active.
- All guarded activities activate deactivated guards.

- When an activity instance is guarded the guard is on watch from launching such an activity instance till its termination.
- A guard stays active between immediately succeeding guarded activity instances.
- When there is no reason to stay active, a guard is deactivated.

In the sample workflow in Fig. 3 nodes corresponding to guarded activities are colored. An activity A belongs to the activation set of a guard G. The workflow engine will activate a guard instance G before the actual start of an instance of the activity A. A has a guarded successor B. When the instance of A completes, the guard instance will be kept active for the execution time of B and deactivated, when the instance of B completes.

This policy is quite straightforward for activating a data guard, but a bit more difficult for deactivation. According to the policy, a guard instance is deactivated, when a guarded activity instance is completed and none of its immediate successors is guarded and there are no parallel branches with launched and unfinished guarded activities, or finished guarded activities with guarded successors forthcoming (par-join).

The deactivation of an active guard instance is a twofold problem. First, when completing a guarded activity instance, the workflow engine has to check with the workflow definition whether there is (possibly) a guarded successor. Second, the workflow engine has to keep track of parallel guarded activity instances. We analyze the problem of deactivation of an active data guard instance in detail in Sec. 3.3.

An example workflow definition is presented in Fig. 4. Only an activity 'Attach photo' belongs to the activation set of a guard G1 and can activate a guard instance for the first time. 'Attach photo' is the first guarded activity following the CS1 node in a path where a claim value is less than 500. Therefore, an instance of G1 can be activated only if the claim value was less than 500. Two sample instance types of this definition are presented in Fig. 5 In the first instance type (a), the workflow engine activates an instance of the guard G1, before the actual start of an instance of the activity 'Attach photo'. In the other instance type (b), the activity 'Check police report' is guarded in the definition, but it does not belong to the activation set of the guard G1 and the guard instance was not activated earlier, i.e. it is still inactive. Therefore, the workflow engine does not activate the guard instance. Observe that in this case the claim value is not less than 500 and a guard instance would be violated immediately after activation.

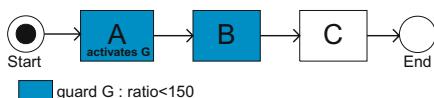
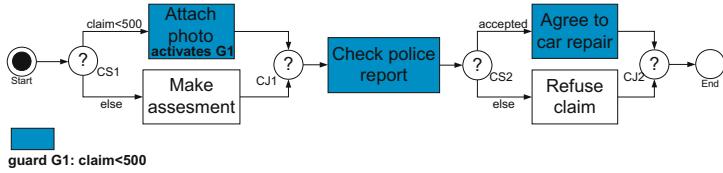
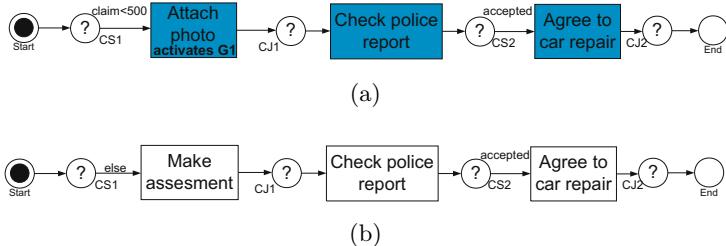


Fig. 3. Workflow graph with a workflow data guard

**Fig. 4.** Workflow data guard with a non empty activation set**Fig. 5.** Sample instance types of a workflow definition in Fig. 4

A condition violation of an active workflow data guard instance raises an exception. The system may react to an exception by passing the exception to an external system or authority (e.g. inform the workflow administrator), starting a complex recovery procedure, finding a way to overcome the problem and continue processing, etc. Different methods of handling exceptions in workflows can be found in the literature, e.g. [2, 8, 11]. These are not part of this paper.

The workflow data guards can guarantee that, during the execution of guarded activities, the guard conditions defined on data used within a workflow will not be violated unnoticed. This enables data guards to be used to:

- describe additional business data constraints in a workflow definition, e.g. a data guard can check whether the stock price stays within a given limit.
- ensure the correctness of data in selected parts of a workflow.
- react to changes of data relevant to the correctness of a workflow.

In particular, workflow data guards can be used to resolve problems with data modifications invalidating past control flow decisions.

3 Activation and Deactivation of Workflow Data Guards

Based on a workflow definition, the workflow engine dynamically activates and deactivates workflow data guards in a running workflow instance. Before we can describe the activation and deactivation mechanism in detail, we need to introduce some basic properties of guards.

3.1 Definitions and Basic Properties

We define the following operations and predicates on the nodes of a workflow graph wg . The graph wg can be either a definition graph or an instance type graph. $N.type$ yields the type of the node N (i.e. one of the following: activity, cond-split, cond-join, par-split, par-join). A predicate $succ(N, M)_{wg}$ is true if M is an adjacent successor of the node N in the graph wg . By analogy, $pred(N, M)_{wg}$ is true if M is an adjacent predecessor of N in wg .

For the activity nodes N we define the predicate $guarded(N, G)_{wg}$ which is true if in a workflow graph wg the activity N is guarded by the guard G :

$$guarded(N, G)_{wg} \Leftrightarrow N.type = \text{activity} \wedge N \text{ is guarded by the guard } G$$

We define for a guarded activity N and a workflow data guard G a predicate $firstActivates(N, G)_{def}$ which is true if a start of an instance of the activity N can activate an inactive instance of the guard G for the *first time*. A workflow designer marks in a workflow definition def for which activities this predicate is true. We define in a workflow definition graph def an activation set of a workflow data guard G as:

$$\text{activationSet}(G)_{def} = \{N : guarded(N, G)_{def} \wedge firstActivates(N, G)_{def}\}$$

Direct activity successor: The predicate $succ'(N, M)_{wg}$ is true in a workflow graph wg if M is an activity which follows the node N , and M is an adjacent successor of N , or in at least one path between N and M there are only control nodes, i.e. in one path M is the first activity after the node N . We say that M is a *direct activity successor* of N :

$$\begin{aligned} succ'(N, M)_{wg} \Leftrightarrow M.type &= \text{activity} \wedge \left(\begin{aligned} &succ(N, M)_{wg} \\ &\vee (\exists V : succ(N, V)_{wg} \wedge V.type \neq \text{activity} \wedge succ'(V, M)_{wg}) \end{aligned} \right) \end{aligned}$$

The predicate $guardedSucc(N, G)_{wg}$ is true in a workflow graph wg , if N has a direct activity successor, which is guarded by G :

$$guardedSucc(N, G)_{wg} \Leftrightarrow \exists M : succ'(N, M)_{wg} \wedge guarded(M, G)_{wg}$$

For each node N in a workflow graph wg we define a set of its direct activity successors, which are guarded by G :

$$guardedSuccSet(N, G)_{wg} = \{M : succ'(N, M)_{wg} \wedge guarded(M, G)_{wg}\}$$

For each node N in a workflow graph wg we define a number of outgoing paths which contain guarded direct activity successors of N :

$$\#\text{guardedSucc}(N, G)_{wg} = \left| \left\{ M : \begin{aligned} &succ(N, M)_{wg} \\ &\wedge ((M.type = \text{activity} \wedge guarded(M, G)_{wg}) \\ &\vee (M.type \neq \text{activity} \wedge guardedSucc(M, G)_{wg})) \end{aligned} \right\} \right|$$

By analogy we describe direct activity predecessors of a node N in a workflow graph wg :

$$\begin{aligned} \text{pred}'(N, M)_{wg} \Leftrightarrow & M.\text{type} = \text{activity} \wedge \left(\text{pred}(N, M)_{wg} \right. \\ & \left. \vee (\exists V : \text{pred}(N, V)_{wg} \wedge V.\text{type} \neq \text{activity} \wedge \text{pred}'(V, M)_{wg}) \right) \end{aligned}$$

$$\text{guardedPred}(N, G)_{wg} \Leftrightarrow \exists M : \text{pred}'(N, M)_{wg} \wedge \text{guarded}(M, G)_{wg}$$

$$\text{guardedPredSet}(N, G)_{wg} = \{M : \text{pred}'(N, M)_{wg} \wedge \text{guarded}(M, G)_{wg}\}$$

It is easy to see that a set of guarded direct activity successors of a node N in a workflow instance type graph $inst$ is a subset of guarded direct activity successors of the node N in a corresponding workflow definition graph def , because each workflow instance type graph $inst$ is a subgraph of its workflow definition graph def . The following two properties are a consequence of this fact.

Proposition 1. *If a node N does not have any guarded direct activity successors in the workflow definition graph, then it does not have any such successor in any instance type graph of this definition.*

Proposition 2. *If a node N has a guarded direct activity successor in the workflow definition graph, then it possibly may not have any guarded direct activity successor in a particular instance type graph of this definition.*

Guarded Path: A guarded path is a path in a *workflow instance* type graph, which contains only instances of the activities guarded by the same guard and possibly some control nodes. A path in a workflow graph is a sequence of nodes N_0, N_1, \dots, N_m such that for $j = 1, \dots, m$, the nodes N_{j-1} and N_j are adjacent. Two guarded paths are parallel, if at least one activity instance from one path can be executed in parallel with any activity instance from the other guarded path.

An instance of the activity N is the *last* in a path guarded by G, if it is guarded itself and does not have in an instance type graph $inst$ any direct guarded activity successors, i.e. $\text{guarded}(N, G)_{inst} \wedge \neg \text{guardedSucc}(N, G)_{inst}$.

By analogy the *first* activity instance in a guarded path is described as follows: $\text{guarded}(N, G)_{inst} \wedge \neg \text{guardedPred}(N, G)_{inst}$.

For example, 'Attach photo' in Fig. 5(a) is the first guarded activity instance in a guarded path, 'Agree to car repair' is the last guarded activity instance in a guarded path, and the guarded path consists of ('Attach photo', CJ1, 'Check police report', CS2, 'Agree to car repair').

3.2 Activation of a Workflow Data Guard

The workflow engine activates instances of workflow data guards according to the workflow definition and the current state of processing. Instances of activities

marked as guarded by a guard G in the workflow definition will be guarded at runtime only when an instance of G is active. The instance of G can be activated by a start of an instance of an activity N marked as guarded by G in the workflow definition only if one of the following holds:

- the activity N belongs to the activation set of G, i.e. $firstActivates(N, G)_{def}$ is true,
- the instance of G is deactivated, i.e. it has already been activated at least once.

This simple and powerful activation mechanism allows the designer to choose very selectively activities and paths where constraints are checked to achieve both the desired correctness guarantees and minimize the performance implications.

3.3 Deactivation of a Workflow Data Guard

To deactivate active instances of workflow data guards, we have to solve the following problems at runtime:

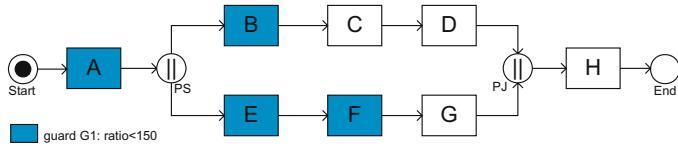
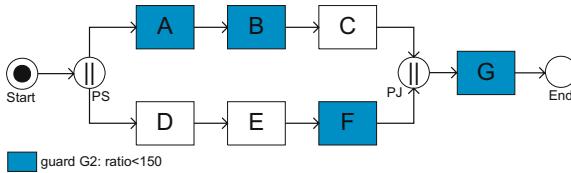
- Keeping track of several guarded paths running in parallel and providing communication between them.
- Detecting that a given guarded path has completed.

Parallel Guarded Paths: An active instance of a workflow data guard can only be deactivated if at a given point of time there are no running parallel paths guarded by this instance. Parallel guarded paths are generated by par-splits:

- A guarded path may begin before a par-split node PS and may continue after the node PS (as in Fig. 6).
- A guarded path may begin after a par-split node PS and may continue after the corresponding par-join node PJ (as in Fig. 7).

In the former case, a guarded path started before a par-split node PS may split in this node into several parallel guarded paths. The maximal number of guarded paths possibly outgoing from the par-split node PS at runtime is described in the workflow definition graph by $\#guardedSucc(PS, G)_{def}$. In the latter case, parallel guarded paths are started independently of each other.

For example, in the graph in Fig. 6 there are two parallel guarded paths: (A, PS, B) and (A, PS, E, F). When an instance of the guarded activity A starts, the corresponding instance of guard G1 is activated. In the par-split node PS one guarded path splits into two guarded paths. The guard instance must stay active till both guarded paths are completed. In the graph in Fig. 7 there are also two parallel guarded paths: (A, B) and (F, PJ, G). The par-join node PJ synchronizes its parallel predecessors. An activity instance G cannot be started before both activity instances C and F have completed. If an activity instance F completes before an activity instance C completes, then the guard instance will be kept active for the processing time of C and deactivated when G completes. If B

**Fig. 6.** Workflow graph with parallel guarded paths**Fig. 7.** Workflow graph with parallel guarded paths

completes before F starts, then the instance of the guard G2 can be deactivated, and will be activated again when the instance of F starts.

Completion of a Guarded Path: Guarded paths are defined in workflow instance type graphs which are constructed dynamically at runtime by the workflow engine according to the workflow definition and control flow decisions. At a given point of workflow processing the workflow engine does not have the complete instance type graph. When a guarded activity instance completes, it is sometimes impossible to decide using only a workflow definition, which (possibly guarded) successor of this activity will be actually started. To detect the last guarded activity instance in a guarded path we have to analyze the following cases:

- A guarded instance of an activity N does not have any guarded direct activity successor in any instance type of a given workflow definition, i.e. each instance of N is always the last activity instance in a guarded path.
- A guarded instance of an activity N may have a guarded direct activity successor in one instance type of a given workflow definition, or may not have any guarded direct activity successor in the other instance type of this definition. This is implied by Proposition 2. When N completes, the workflow engine cannot decide at runtime, whether the instance of N is the last in the guarded path, unless a successor of N is started.

In the first case, we use Proposition 1. If an activity N guarded by G does not have any guarded direct activity successors in the workflow definition graph, i.e. $\neg \text{guardedSucc}(N, G)_{\text{def}}$, then its instances do not have any guarded direct activity successors in any of the instance types. Each guarded instance of activity N is therefore the last activity instance in a guarded path.

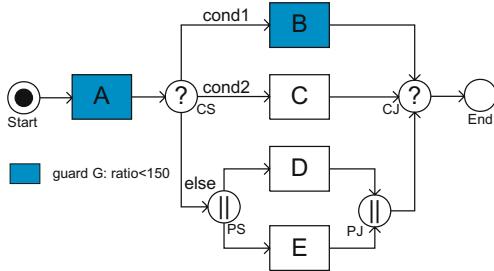


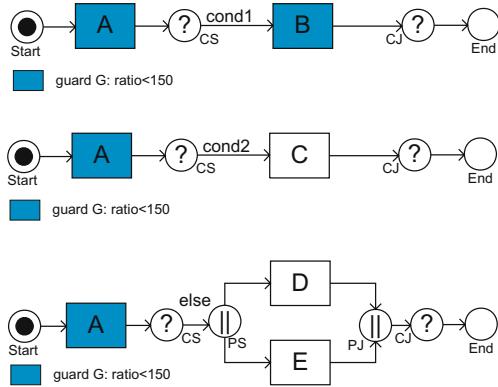
Fig. 8. Workflow definition, where it is unknown if an instance of A will have a guarded direct activity successor at runtime

The other case is much more interesting. A guarded activity instance N may have a guarded direct activity successor in one instance type graph or may not have such a successor in the other instance type graph, if *all of the following* hold in a workflow definition graph def :

- For the guarded activity N it holds $guardedSucc(N, G)_{def}$, i.e. N has guarded direct activity successors. For example, an activity A in a workflow definition in Fig. 8 has a guarded direct activity successor B.
- There exists a path in the workflow definition, where between the guarded activity N and one of its guarded direct activity successors is a cond-split node. In our example in Fig. 8 there is a cond-split CS between A and B.
- At least one of the adjacent successors of this cond-split node is neither a guarded activity nor a control node which has guarded direct activity successors. In the example in Fig. 8 we have both cases. The activity C is not guarded and the par-split node PS does not have any guarded direct activity successors.

The workflow definition in Fig. 8 can have instances in one of three workflow instance types presented in Fig. 9. In each of these instance types, the workflow engine cannot deactivate a guard instance G when an activity instance A completes. This operation must be postponed, until a control flow decision is made in the node CS and one of its adjacent successors is going to be started. If a selected successor of CS is the activity C or the par-split PS, then the guard instance must be deactivated before the actual start of this successor.

We can identify nodes similar to the activity C and par-split PS in Fig. 8. Such a node N is an adjacent successor of a cond-split node CS in a workflow definition. The node CS has both a guarded direct activity successor and a guarded direct activity predecessor, i.e. a guarded path starts before CS and may be continued after CS in one of the possible workflow instance types. But the node N itself is neither a guarded activity nor a control node with guarded direct activity successors. When the node N is going to be instantiated at runtime, the guarded path has finished just before N . We define the following predicate to describe such a node N in a workflow definition graph def :

**Fig. 9.** Instance types of a workflow definition if Fig. 8

$$\begin{aligned}
 & \text{endsGuardedPath}(N, G)_{\text{def}} \Leftrightarrow \\
 & \quad \text{guardedPred}(N, G)_{\text{def}} \wedge \left((N.\text{type} = \text{activity} \wedge \neg \text{guarded}(N, G)_{\text{def}}) \right. \\
 & \quad \left. \vee (N.\text{type} \neq \text{activity} \wedge \neg \text{guardedSucc}(N, G)_{\text{def}}) \right) \\
 & \quad \wedge \exists CS : \left(\text{pred}(N, CS)_{\text{def}} \wedge CS.\text{type} = \text{cond-split} \right. \\
 & \quad \left. \wedge \text{guardedSucc}(CS, G)_{\text{def}} \wedge \text{guardedPred}(CS, G)_{\text{def}} \right)
 \end{aligned}$$

3.4 Tracking the Guarded Paths

We use tokens to detect a completion of a guarded path and provide communication between guarded paths running in parallel. With each instance of the workflow data guard G we associate at runtime a multi-set of colored tokens: $\text{tokens}(G)$. A color is given to a token by a unique identifier of a guarded activity. Each token in $\text{tokens}(G)$ corresponds to one path guarded by G and being

Table 1. Operations on a multi-set $\text{tokens}(G)$

Operation	Operation semantics
<code>isEmpty()</code>	Returns true, if the multi-set does not contain any element
<code>contains(element)</code>	Returns true, if the specified element is in the multi-set
<code>add(set)</code>	Adds one copy of each element of the input set
<code>add(set, number)</code>	Adds the specified number of copies of each element of the input set
<code>remove(set)</code>	Removes one copy of each element of the input set

currently processed at runtime. A guard instance is initiated at runtime as inactive and without any tokens. The guard instance is active, when it contains a token and can be deactivated only when there are no more tokens.

A multi-set is an unordered collection of elements that may have duplicates, i.e. $\text{tokens}(G)$ can contain multiple identifiers of the same guarded activity. The allowed operations on $\text{tokens}(G)$ are defined in Table 1.

The workflow engine adds a token corresponding to N into the multi-set $\text{tokens}(G)$ before the actual start of a guarded activity instance N. Before the actual completion of the activity instance N, the workflow engine checks in the workflow definition, whether this activity instance may have a guarded direct activity successor, i.e. if $\text{guardedSucc}(N, G)_{\text{def}}$ is true. If this is the case, the token N is left in $\text{tokens}(G)$ to indicate a guarded path being processed. Otherwise the engine removes the token N from $\text{tokens}(G)$.

If a guarded path has started before a par-split node PS and splits in this node into several guarded paths, then the workflow engine adds into $\text{tokens}(G)$ ($\#\text{guardedSucc}(PS, G)_{\text{def}} - 1$) copies of each token actually left in $\text{tokens}(G)$ after the direct guarded predecessors of PS. Each copy of a token added into $\text{tokens}(G)$ indicates one parallel guarded path being currently processed.

While processing a guarded path, the workflow engine replaces tokens left after actual guarded direct activity predecessors of a guarded activity M with a token corresponding to M itself. The workflow engine removes from $\text{tokens}(G)$ one copy of each token corresponding to instances in a set $\text{guardedPredSet}(M, G)_{\text{inst}}$ and adds only one copy of a token corresponding to M.

The workflow engine has to ensure that, after the completion of the last guarded activity instance in a guarded path, no token corresponding to this path is left in $\text{tokens}(G)$. A token corresponding to N is left in $\text{tokens}(G)$ if $\text{guardedSucc}(N, G)_{\text{def}}$ is true. If after a guarded instance of the activity N, the workflow engine is going to instantiate a node M, for which the predicate $\text{endsGuardedPath}(M, G)_{\text{def}}$ is true, then a given guarded path has finished before the node M. In this case, the workflow engine removes from $\text{tokens}(G)$ one copy of each token corresponding to activities in $\text{guardedPredSet}(M, G)_{\text{inst}}$.

3.5 Workflow Metamodel and Algorithms

We use a workflow metamodel to describe workflow definitions, workflow instances, workflow data and workflow data guards. The complete metamodel can be found in [10]. Because of lack of space we present in this paper a simplified view on our metamodel. In this view both workflow definitions and workflow instances are seen as graphs as presented in Fig. 10.

A workflow definition graph consists of nodes. Each node has a unique identifier and a type (activity, cond-split, par-split etc.). A node can have adjacent successors and predecessors. The workflow definition may also contain workflow data guards. A guard is put on one or more activity nodes. Some of the guarded activities can activate at runtime an inactive instance of a given guard for the first time. A set of such activities is returned by a method $\text{activationSet}()$ of the guard. The methods of a node in a workflow definition are described in Table 2.

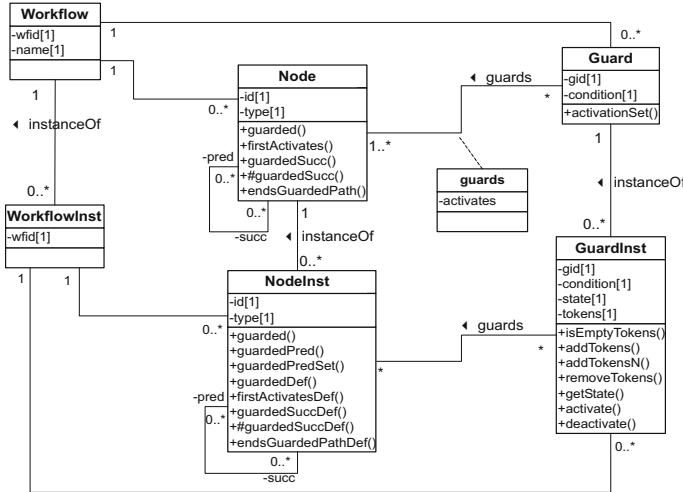


Fig. 10. View on a general workflow metamodel incorporating workflow data guards concept

Table 2. Guard related methods of a node N

Method	Description
N.guarded(G)	Returns true, if the predicate $guarded(N, G)_{def}$ is true
N.firstActivates(G)	Returns true, if the predicate $firstActivates(N, G)_{def}$ is true
N.guardedSucc(G)	Returns true, if the predicate $guardedSucc(N, G)_{def}$ is true
N.#guardedSucc(G)	Returns a number given by $\#guardedSucc(N, G)_{def}$
N.endsGuardedPath(G)	Returns true, if the predicate $endsGuardedPath(N, G)_{def}$ is true

A workflow instance graph is a subgraph of a workflow definition graph. The instance graph consists of node instances, which can have predecessors and successors. The type of a node instance is the same as the type of the corresponding node in the workflow definition. Node instances may be guarded by guard instances. The methods of a node instance are described in Table 3. The methods of a guard instance are described in Table 4.

The workflow engine has methods which interpret the workflow definition, start and complete activity instances, and make the control flow decisions [4]. The additional logic needed for activating and deactivating the workflow data guard instances is presented in Algorithms 1 and 2.

The method *checkGuardsByStart* in Algorithm 1 is used by the workflow engine before the actual start of each node instance. If a started node is an instance of an activity guarded by a guard G in the workflow definition (line 4),

Table 3. Guard related methods of a node instance N

Method	Description
N.guarded(G)	Returns true, if a predicate $guarded(N, G)_{inst}$ is true
N.guardedPred(G)	Returns true, if a predicate $guardedPred(N, G)_{inst}$ is true
N.guardedPredSet(G)	Returns a set of unique identifiers of node instances from the set $guardedPredSet(N, G)_{inst}$
N.guardedDef(G)	Returns the same result as a method of the corresponding node in the workflow definition
N.firstActivatesDef(G)	
N.#guardedSuccDef(G)	
N.endsGuardedPathDef(G)	
N.guardedSuccDef(G)	

Table 4. Methods of a workflow data guard instance G

Method	Description
G.getState()	Returns the actual state of the instance G. Possible state values: <i>inactive</i> , <i>active</i> , <i>deactivated</i>
G.activate()	Activates the guard instance. Changes the state of G into <i>active</i>
G.deactivate()	Deactivates the guard instance. Changes the state of G into <i>deactivated</i>
G.isEmptyTokens()	Same as a corresponding operation on a multi-set $tokens(G)$ described in Table 1
G.addTokens(set)	
G.addTokensN(set,number)	
G.removeTokens(set)	

the workflow engine checks if a token corresponding to this activity instance can be added into tokens of the guard instance (line 5). If this is the case, the token is added (line 6) and the guard instance is activated (line 8). If this guarded activity instance has guarded direct activity predecessors in the workflow instance graph, then the workflow engine removes from the guard instance the tokens left after those predecessors (line 11).

If a started node is an instance of a par-split node in which one guarded path splits into several guarded paths in the workflow definition (line 14), then the workflow engine adds into the guard instance copies of each token left by the guarded direct activity predecessors. The number of added copies depends on a number of guarded paths possibly outgoing from this par-split node in the workflow definition (lines 15-16).

If a started node instance is an instance of a node, which starts a non guarded path in the workflow definition graph (line 17), the workflow engine removes tokens left after the guarded direct activity predecessors of this node instance. The guard can be deactivated, if afterwards there are no more tokens (line 20).

The method *checkGuardsByCompletion* in Algorithm 2 is used by the workflow engine before the actual completion of each guarded activity instance.

Algorithm 1. Check guards before the actual start of a node instance in a workflow instance

```

1: Procedure checkGuardsByStart(nodeInst N)
2: workflowInstance := N.getWorkflowInstance()
3: for all guard instances G in workflowInstance do
4:   if N.type = activity  $\wedge$  N.guardedDef(G) then
5:     if N.firstActivatesDef(G)  $\vee$  G.getState()  $\neq$  inactive then
6:       G.addTokens({N.id})
7:       if G.getState()  $\neq$  active then
8:         G.activate()
9:       end if
10:      if N.guardedPred(G) then
11:        G.removeTokens(N.guardedPredSet(G))
12:      end if
13:    end if
14:   else if N.type = par-split  $\wedge$  N.guardedPred(G)
15:      $\wedge$  N.#guardedSuccDef(G)  $>$  1 then
16:     number := N.#guardedSuccDef(G) - 1
17:     G.addTokensN(N.guardedPredSet(G), number)
18:   else if N.endsGuardedPathDef(G) then
19:     G.removeTokens(N.guardedPredSet(G))
20:     if G.isEmptyTokens() then
21:       G.deactivate()
22:     end if
23:   end if
24: end for

```

Algorithm 2. Checks guards before the actual completion of an activity instance in a workflow instance

```

1: Procedure checkGuardsByCompletion(nodeInst N)
2: if N.type = activity then
3:   workflowInstance:= N.getWorkflowInstance()
4:   for all guard instances G in workflowInstance do
5:     if N.guarded(G)  $\wedge$   $\neg$ N.guardedSuccDef(G) then
6:       G.removeTokens({N.id})
7:       if G.isEmptyTokens() then
8:         G.deactivate()
9:       end if
10:     end if
11:   end for
12: end if

```

If this guarded activity instance does not have any guarded direct activity successors in the workflow definition, then it is definitely the last activity instance in a given guarded path. The engine removes from the guard instance a token corresponding to this activity instance (line 6). The engine deactivates the guard instance, if afterwards there are no more tokens (line 8).

The presented algorithms were implemented in Java and integrated with a mini workflow engine developed at the University of Klagenfurt. A number of simulations showed that guard instances were correctly activated and deactivated at runtime according to a given workflow definition.

4 Checking the Condition of an Active Workflow Data Guard Instance

The condition of a workflow data guard is defined on data used within a given workflow definition. At runtime the workflow engine checks this condition on data instances. The condition of a data guard instance is checked only when the guard instance is active. This condition cannot be violated during the whole activation time of this guard instance. The workflow engine can use one of two ways of checking the condition. First, it can use the active capabilities (e.g. triggers) of a data source, where the workflow data are stored. Alternatively, the workflow engine can actively check the condition itself.

A database trigger, used by the workflow engine to check the conditions of active guard instances, signals a violation of these conditions. The trigger is defined per a workflow data guard, not per a workflow data guard instance. If there is no instance of a given workflow data guard being active, then the corresponding trigger can be deactivated. The condition of the guard is used to define the condition of the trigger. The trigger uses additional information about each guard instance created. This is a collection of triples, which contain: an identifier of a workflow data guard instance, a state of this instance and the identification of data guarded by this instance (e.g. primary key or document identification). When data used to define the condition of an active guard instance are changed in a way that violates this condition, the trigger signals it to the workflow engine. The engine receives the identification of the violated guard instance.

In an alternative approach, the workflow engine actively checks the conditions of active guard instances in a given workflow instance. The condition of an active guard instance is checked at each state transition of each activity instance in the workflow. The condition is checked even if a given activity instance is not guarded. This method can be used when it is not possible to define the triggers.

5 Related Work

The influence of data aspects on correctness of workflows has not yet received much interest in the literature. In most cases the authors analyzed the correctness of data flows in workflow specifications, e.g. in [13] are listed possible problems with data flow in workflow models.

Many WfMSs allow pre- and postconditions to activities (e.g. [4, 12]). In most cases these pre- and postconditions are specified in an activity definition and apply to all instances of a given activity. This may reduce the reusability of such activities. Our workflow data guards do not reduce the reusability of predefined activities. The same activity can be used many times, once guarded and once not.

Active databases and production rules (triggers) [3] were used for workflow enactment, as a promising operational model for workflows. Because an extensive use of many triggers could lead to a degeneration in system performance, triggers are mainly used for detecting and signaling events generated by database modifications (e.g. in WIDE [9]).

The importance of incorporating exception handling into WfMSs has been identified by researchers (e.g. WAMO [7], WIDE [2]). WAMO and its extensions [8] offered advanced mechanisms for handling exceptions in workflows.

In ConTracts [14] invariants were used to protect data needed for compensation of successfully finished steps. They ensured that the execution of compensation actions would be always allowed.

6 Conclusions

Workflow data guards contribute to an improved treatment of data aspects in workflows considering that workflows work on data which are shared between several applications and are frequently maintained outside of the workflow management system. They provide a powerful mechanism for workflow designers for recognizing changes to data relevant for a workflow and to react to these changes to ensure the correct execution of a workflow. In particular, it is possible to monitor data that was influential in making flow decisions and thus allows the workflow designer to achieve some transactional qualities of workflow execution without the need to hand-code activities which have pure technical purpose of checking stability of relevant data. Moreover, data guards can also be used to specify additional constraints (e.g. expected invariants of activities) and have them automatically checked by the workflow management system.

The data guard mechanism we presented is designed to be generally enough to be easily applicable to any full-blocked workflow model, as demonstrated in our prototypical implementation. The elaborated activation and deactivation mechanism gives the designer the possibility to achieve the desired properties of the workflow execution without unnecessarily decreasing the performance with unnecessary checks. More details on our data guard implementation, like a full metamodel, guard notations in our workflow language WDL-X, etc. can be found in [10].

References

1. Christoph Bussler. Has Workflow Lost Sight of Dataflow?, 1999. High Performance Transaction System Workshop 1999.
2. Fabio Casati, Stefano Ceri, Stefano Paraboschi, and Giuseppe Pozzi. Specification and Implementation of Exceptions in Workflow Management Systems. *ACM Trans. Database Syst.*, 24(3):405–451, 1999.
3. Umeshwar Dayal, Eric Hanson, and Jennifer Widom. Active database systems. In Won Kim, editor, *Modern database systems: the object model, interoperability, and beyond*, pages 434–456. ACM Press/Addison-Wesley Publishing Co., 1995.

4. Johann Eder, Herbert Groiss, and Walter Liebhart. The Workflow Management System Panta Rhei. In A. Dogac, L. Kalinichenko, T. Öszu, and A. Sheth, editors, *Workflow Management Systems and Interoperability*. Springer-Verlag, 1998.
5. Johann Eder and Marek Lehmann. Uniform Access to Data in Workflows. In Kurt Bauknecht, Martin Bichler, and Birgit Pröll, editors, *Proceedings of the 5th International Conference on E-Commerce and Web Technologies, EC-Web 2004*, volume 3182 of *LNCS*, pages 66–75, Zaragoza, Spain, August/September 2004. Springer-Verlag.
6. Johann Eder and Marek Lehmann. Synchronizing Copies of External Data in Workflow Management Systems. In Oscar Pastor and João Falcão e Cunha, editors, *Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005. Proceedings*, volume 3520 of *LNCS*, pages 248–261. Springer Varlag, 2005.
7. Johann Eder and Walter Liebhart. The Workflow Activity Model WAMO. In *Proc. of the 3rd Int. Conf. on Cooperative Information Systems (CoopIS)*, 1995.
8. Johann Eder and Walter Liebhart. Workflow Recovery. In *First IFCIS Intl. Conf. on Cooperative Information Systems (CoopIS'96)*, pages 124–134. IEEE Computer Society Press, June 1996.
9. Paul Grefen, Barbara Pernici, and Gabriel Sánchez, editors. *Database Support for Workflow Management. The WIDE Project*. Kluwer Academic Publishers, 1999.
10. Marek Lehmann. *Data Access in Workflow Management Systems*. PhD thesis, University of Klagenfurt, 2005.
11. Zongwei Luo, Amit Sheth, Krys Kochut, and John Miller. Exception Handling in Workflow Systems. *Applied Intelligence*, 13(2):125–147, 2000.
12. Nick Russell, Arthur H. M. ter Hofstede, David Edmond, and W.M.P. van der Aalst. Workflow Data Patterns. Technical Report FIT-TR-2004-01, Queensland University of Technology, Brisbane, Australia, April 2004.
13. Shazia Sadiq, Maria Orlowska, Wasim Sadiq, and Cameron Foulger. Data Flow and Validation in Workflow Modelling. In *CRPIT '27: Proceedings of the fifteenth conference on Australasian database*, pages 207–214. Australian Computer Society, Inc., 2004.
14. Helmut Wächter and Andreas Reuter. The ConTract Model. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 219–263. Morgan Kaufmann Publishers Inc., 1992.

Consistency Between e^3 -value Models and Activity Diagrams in a Multi-perspective Development Method

Zlatko Zlatev and Andreas Wombacher

University of Twente, Department of Computer Science, Information System Group,
P.O. Box 217, 7500 AE Enschede, The Netherlands
Z.V.Zlatev@ewi.utwente.nl
A.Wombacher@utwente.nl

Abstract. Multi-perspective approaches to analysis and design of businesses information systems are used to manage the complexity of the development process. A perspective contains a partial specification of the system from a particular stakeholder's standpoint. This separation of concerns leads to potential inconsistencies between specifications from different perspectives, resulting in non-implementable systems. In this paper, a consistency relationship between the *economic value* and *business processes* perspectives of a design framework for networked businesses is proposed based on an equivalence of a common semantic model.

1 Introduction

The development of an information system to support a business is a complex process. Many stakeholders with distinctive interests are involved in the alignment of information system capabilities and business objectives. A standard approach to manage the complexity of such a process is the adoption of a multi-perspective development method, where responsibilities for design and analysis of distinctive aspects of the system are localized in separate perspectives. We use a framework with three perspectives, including: (i) *economic value*, in which we model the creation of value among networked businesses and analyze the incentives for them to take part in such a network; (ii) *business processes*, in which we model the coordination of activities realizing the exchanges of economic values; and (iii) *application communication*, in which we model the data exchange among the components of the information system that supports the business.

A multi-perspective approach presupposes a decentralized development process, the benefits of which come at a price of potential inconsistencies between models from different perspectives. In particular, specifications are inconsistent if it is not possible to build a single system that correctly implements the specification of each perspective. As a consequence, specifications are consistent, if an implementation of the specifications exists. Therefore, specifications need to be checked for consistency to identify required changes in the final design.

The inconsistencies occur due to different design methodologies, opposing stakeholders' goals, conflicting knowledge and incompatible modeling notations. In particular, inconsistencies emerge from the redundant information in different

models; i.e., perspectives overlap in their responsibilities for modeling certain aspects of the system. Therefore, we base our consistency check on the common concepts and relations of two modeling notations. We call such a limited modeling notation a *reduced model* as it contains only constructs present in both models. We define consistency between pairs of models, which is a necessary condition for global consistency. The common concepts and relations of the three models are too limited, with regard to the precision of a consistency check involving all three perspectives, resulting in unusable consistency decisions.

We operationalize our consistency definition by transforming the specifications from different perspectives to reduced models and by defining an equivalence relationship between reduced models. Hence, a specification may represent different scenarios and because the concepts of separating these scenarios are incomparable in the different perspectives, a single reduced model represents a single alternative scenario of the original specification. (An example of an alternative in a process model is an execution sequence without decision point.) Thus, a single specification may produce more than one reduced model and the equivalence relationship is defined between sets of reduced models.

To illustrate the proposed approach, we use exemplary modeling notations, although, other notations could have been used. We represent the value perspective with the e^3 -value modeling notation [7], the business process perspective with UML Activity diagram [10 pages 3-155—3-169], and the application communication perspective with, e.g., communication diagram [14 pages 201—211] or UML Component diagram [10 pages 3-169—3-171]. This selection of modeling notations impacts the particular common concepts but is invariant to the proposed approach based on reduced models.

The contribution of this paper is the definition of consistency between e^3 -value models and activity diagrams. Further, we discuss applying the presented approach to the remaining two consistency relationships: e^3 -value model and communication diagram, and activity diagram and communication diagram.

The paper is structured as follows: In Section 2, we present an example business case and its models from value and process perspectives. Additionally, we provide an intuitive consistency definition which we use later as a validation criterion. In Section 3, we define the reduced model between the e^3 -value notation and UML Activity diagram and introduce our consistency definition. Further in Section 4, we discuss the impact of model granularity on our consistency definition. Section 5 validates the proposed consistency definition by a comparison with the intuitive consistency definition. The last three sections discuss implications for the application communication perspective, related work, and conclusion and future work.

2 Example

We consider a business case with the following businesses taking part: a buyer, a seller, and a shipping company. The seller has a shop and a warehouse at two different locations. It can directly sell products to customers only from the shop. If a product is purchased that is not present in the shop then a delivery from the warehouse must be made. A shipping company is paid to arrange the logistics to the

buyer's home. The seller processes two payment methods: (1) in a case of an off-the-shelf product, the seller requires immediate payment in cash; (2) in a case of a purchase from the warehouse, the seller allows late payment by, e.g., a bank transfer.

2.1 Economic Value Perspective

We use the e^3 -value modeling notation [7] to represent the value aspect of a business model. Below, we explain the semantics of the concepts we use. We refer to Fig. 1 for the graphical representation on the concepts. An *actor* is an economically independent entity modeled as a rectangle. A *value interface* indicates which value object is available, in return for another value object. It is shown by a rounded box, connected to an actor. A *value exchange* represents that two actors are willing to exchange value objects with each other. It is a prototype for actual trades between actors and is shown by a line. A *value object* represents a value for one or more actors: e.g., services, goods, money, or consumer experiences. Value objects are shown as text.

A *dependency path* represents the internal coordination within actors. It shows via which value interfaces an actor must exchange value objects, given the exchange of objects via another interface of that same actor. A dependency path is a set of dependency nodes and connections. A *dependency node* is a *stimulus* (represented by a bullet), an *OR-fork* (represented by a triangle), or an *end node* (represented by a bull's eye). A stimulus represents a consumer need that triggers the chain of exchanges; an OR-fork represents alternative paths; and an end node represents the model boundary. A *dependency connection* connects dependency nodes and value interfaces, represented by a broken line.

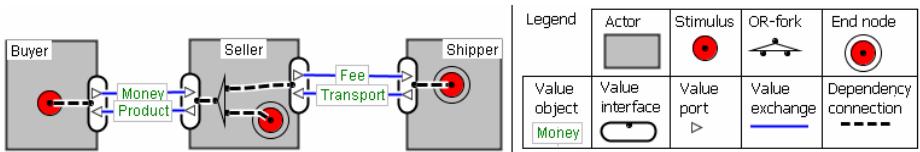


Fig. 1. Value model of the example business case, using e^3 -value modeling notation

Fig. 1 shows a value model of the example business case described above. It contains the three businesses connected by value exchanges. The buyer is willing to give money in return for a product. The two potential exchanges are with the seller, in the middle of the figure, who has an interest in the same value objects. The dependency path, which starts in the buyer, is split at the seller. This is an OR-fork which exemplifies that some products are handed to the buyer immediately while others must be transported by a shipping company. The seller and shipper exchange the value objects Fee and Transport which are paired together reciprocally.

2.2 Business Processes Perspective

The e^3 -value model focuses on the pairing of objects that have economic value for businesses. We now discuss the coordination of activities performed by each business

to achieve the exchange of value objects. We use an UML Activity diagram [10 pages 3-155—3-169] to represent the business processes perspective of our example.

Fig. 2 shows the sequence of actions performed during a purchase of a product. The process starts with the buyer requesting a product. Her order is processed and two outcomes are possible: either the desired product is present in the shop; or the product must be reserved and shipped from the warehouse. These options are represented in Fig. 2 as a choice in the seller's swimlane. In case the path to the left (marked with 1 in the figure) is followed then the product is handed directly and payment in cash is received in return. In the second case (marked with 2 in the figure), a reservation is made. This is followed by two parallel branches which represent the payment of an invoice and the transportation of the product. The latter requires coordination with the logistics provider, which is shown as message exchanges between the seller and shipper swimlanes. The actual delivery of the product is represented in the bottom of Fig. 2 as a message from the shipper swimlane to the buyer.

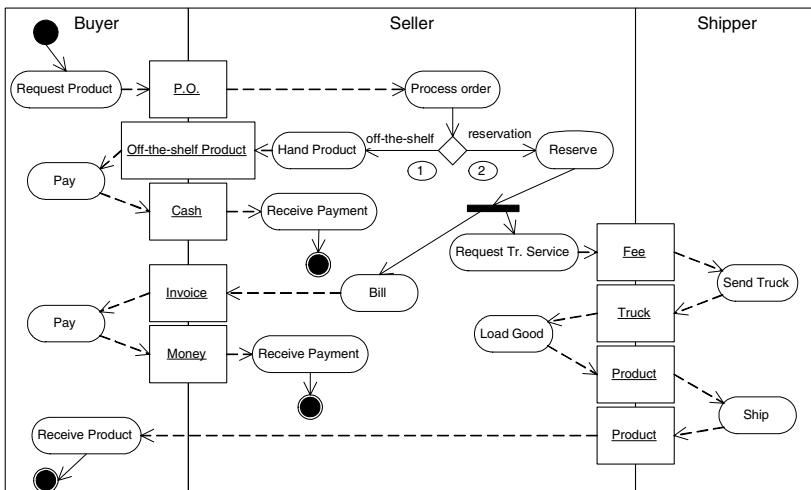


Fig. 2. Process model of the example business case, using UML Activity diagram

2.3 Intuitive Consistency

We consider the presented activity diagram and e^3 -value model to be consistent because there is at least one system implementing both specifications. An activity diagram and an e^3 -value model are consistent if (1) for every alternative dependency path in the value model, an execution sequences exists in the process model such that exactly the product value exchanges described by the path are executed and (2) for every execution sequence in the process model, there exist a dependency path in the value model such that it is possible to bind all exchanged products to all product value exchanges.

The following terms need further clarification:

- an alternative dependency path represents a distinctive scenario of value exchanges in an e^3 -*value* model. A dependency path can include several scenarios, respectively, several alternative dependency paths;
- an execution sequence is a sequence of activities (possibly executed in parallel) in an activity diagram that (1) begins with the start stimulus and ends with termination points and (2) does not contain choices. An activity diagram can include several execution sequences;
- a product value exchange refers to an exchange of good or a service in the e^3 -*value* model.

3 Consistency

The e^3 -*value* model and the activity diagram are not directly comparable. The e^3 -*value* model is based on value exchanges disregarding the order in which they are performed. The activity diagram is based on sequences of object flows disregarding relationships of reciprocal economic value among objects. Thus, we construct a reduced model containing the common concepts and relations of the e^3 -*value* model and the activity diagram to make the two models comparable.

3.1 Reduced Model

The reduced model is used to compare abstractions of the e^3 -*value* model and the activity diagram. To avoid confusion of terminology as both notations use the concept *object*, we refer to objects and object flows in the activity diagram as messages and message exchanges, respectively.

In particular, the reduced model used for consistency checking consists of business units, common value objects, and common value exchanges, where:

- a *business unit* (called *unit* for short) corresponds to an actor from the e^3 -*value* model and a swimlane from the activity diagram. It represents organizational units (grouping of responsibilities) within a business, which are profit and loss responsible but not necessarily legal entities;
- a *common value object* (called *common object* for short) corresponds to a value object from the e^3 -*value* model and a message from the activity diagram. It represents an object of economic value in the e^3 -*value*-model sense which is used for coordination of business activities;
- a *common value exchange* (called *common exchange* for short) corresponds to a value exchange in the e^3 -*value* model and a message exchange in the activity diagram. It represents a bilateral exchange of coordination value objects between profit and loss responsible entities disregarding order, reciprocity and bundling.

Fig. 3 shows the visual notation of the reduced model concepts, where (a) represents a business unit, (b) represents a common value object, and (c) represents a common value exchange.

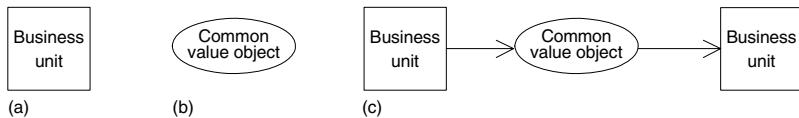


Fig. 3. Visual notation of the reduced model

A reduced model is an explicit representation of a single alternative dependency path in an e^3 -value model and of a single execution sequence in an activity diagram. It contains the value exchanges executed in one possible scenario in a business case.

Common value exchanges are closely related to value exchanges because they are more generic than message exchanges which explicitly represent message ordering. Reciprocity, as contained in the value model, is not considered in the reduced model because there is no corresponding concept in activity diagrams. Alternatives are represented explicitly because OR-forks in e^3 -value models and choices in activity diagrams are not always comparable. This approach of comparing alternatives independently of each other is well known, e.g., from deciding properties of workflow models, which is often based on occurrence graphs derived from Petri nets.

Value objects can be divided into three sub-types, namely goods, services and experiences. The term product refers to both goods and services. We require that products are represented in the reduced model with common objects; whereas, we omit experiences from the reduced model as it is unlikely that these are modeled as message exchanges in an activity diagram.

An exchange of a value object in the e^3 -value-model sense corresponds to a sequence of messages exchanged between two swimlanes in an activity diagram. We will call such a sequence a transaction. Since a sequence of message exchanges does not provide a direction as a value exchange does, we select a single message exchange as a representative of the sequence. The direction of the selected message exchange reflects the direction of the value object exchange. Correspondingly, we map the message of the selected message exchange to a common object.

3.2 Semantic Relationships Between Instances

Besides the conceptual transformation from a value and process model to a reduced model, the instances of the concepts also have to be semantically correlated. The semantic relationship between instances can be one-to-one, one-to-many, and many-to-many. The first two relationship types can be observed in the example described in Section 2 and are covered in this section, while the many-to-many relationship is discussed in Section 4.

The transformation of an e^3 -value model or an activity diagram results in reduced models. In particular, the reduced models must be based on the same set of semantic instances of units and common objects. The existing semantic relationship between instances of actors and swimlanes is represented by two relationships: between an actor and a unit, and between the same unit and a swimlane. Respectively, the semantic relationship between instances of value objects and messages is represented by two relationships: between a value object and a common object, and between the same common object and a message. To restrict the relationships between actors (and

value objects) and swimlanes (and messages) to one-to-one and one-to-many, we allow a unit (and a common object) to take part in at most a single one-to-many relationship.

The instances of a business unit and a common value object in the reduced model are determined by an expert who has knowledge about the instances of the corresponding concepts in the e^3 -value model and the activity diagram. The expert also determines the mapping between the instances, which is captured in transformation tables.

For the business case in our example (Section 2), the mappings of actors from the e^3 -value model and activity diagram to reduced models are listed in Table 1 (a) and Table 1 (b), respectively. Due to the construction of the example, the rows in Table 1 contain the same actor names representing one-to-one relationships. Table 2 (a) lists the mapping between value objects in the e^3 -value model and the common objects in the reduced model. Again, due to the construction of the example this mapping represents a one-to-one relationship. Table 2 (b) lists the mapping between selected messages in the activity diagram and common objects of the reduced model. The mapping in Table 2 (b) contains two one-to-many relationships; i.e., the Money and Cash messages of the activity diagram map to the Money common object of the reduced model, and the Product and Off-the-self product messages of the activity diagram map to the Product common object of the reduced model.

In the following, we describe the transformation of an arbitrary e^3 -value model and activity diagram to their underlying reduced models by means of the example above.

Table 1. Mapping of: (a) actors of the e^3 -value model of Fig. 1 to business units of the reduced model (b) business units of the reduced model to swimlanes of the activity diagram of Fig. 2

(a)		(b)	
e^3 -value model	Reduced model	Reduced model	Activity diagram
Buyer	Buyer	Buyer	Buyer
Seller	Seller	Seller	Seller
Shipper	Shipper	Shipper	Shipper

Table 2. Mapping of: (a) value objects of the e^3 -value model of Fig. 1 to common value objects of the reduced model (b) common value objects of the reduced model to messages of the activity diagram of Fig. 2

(a)		(b)	
e^3 -value model	Reduced model	Reduced model	Activity diagram
Money	Money	Money	Money
Product	Product	Product	Product
Fee	Fee	Fee	Fee
Transport	Transport	Cash	Off-the-shelf product

Further, a notion of consistency based on equivalence of reduced models will be introduced.

3.3 Transformation from an e^3 -value Model to Reduced Models

This section describes a transformation from an e^3 -value model to reduced models. The transformation has three steps. The first step separates possible alternatives in a scenario. The second step selects the actors and value objects to be represented in the reduced model as units and common objects, and builds the transformation tables. The third step transforms actors and value objects to units and common objects.

Step 1: Separate alternatives. This step deals with OR-forks in the dependency path in e^3 -value models. When we encounter an OR-fork, we duplicate the e^3 -value model in accordance to the number of alternatives in the OR-fork. In each copy of the original model, we substitute the OR-fork with a dependency connection to form a single alternative. The other alternatives remain disconnected. This transformation step generates from a single e^3 -value model potentially many e^3 -value models.

The OR-forks are treated, beginning from the start stimuli, consequently in the order of their occurrence along the dependency path. This guarantees that all possible scenarios of value exchanges are captured in individual reduced models. At the end of the step, the exchanges that are not connected in a dependency path are removed from the models. Then, the dependency paths are also removed. The final result of step 1 of the transformation is shown in Fig. 4.

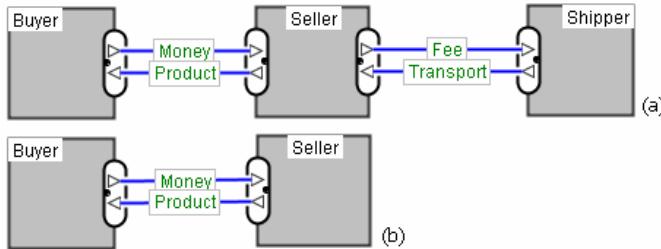


Fig. 4. Final result of transformation step 1

Step 2: Transformation tables. This step classifies the value objects into product and experience types. Product type value objects are entitled for transformation, where the experience type value objects are removed from the e^3 -value model. As a result, actors that exchange only experience type of value objects are isolated and are, therefore, also removed from the model. In the e^3 -value model of our example, all value objects are eligible for translation.

Remaining actors and product type value objects are mapped to business units and common value objects, respectively. With regard to our example, the mappings are represented in the transformation tables Table 1 (a) and Table 2 (a).

Step 3: Generate reduced models. This step transforms each e^3 -value model representing an alternative into a reduced model. In particular, actors and value

objects are transformed into business units and common value objects as specified in the mapping tables (see Table 1 (a) and Table 2 (a)). As a result of this transformation the specific information on reciprocity of value exchanges and on bundling of value objects is omitted. The reduced models, derived from the e^3 -value model (see Fig. 1), are depicted in Fig. 5.

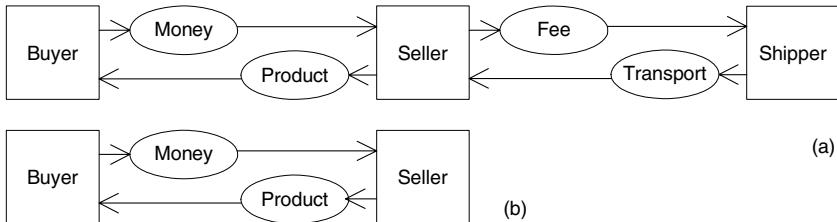


Fig. 5. Reduced models corresponding to the e^3 -value model

3.4 Transformation from an Activity Diagram to Reduced Models

The transformation of an activity diagram to reduced models is performed in three steps. The first step resolves choices in the control flow. The second step identifies sequences of messages, marks single messages as corresponding to value exchanges in the e^3 -value-model sense, and builds transformation tables. The third step transforms swimlanes to units and messages to common objects.

Step 1: Remove choices. This step transforms the activity diagram to a number of models which do not contain choices; i.e. the resulting models do not have conditional branches of execution flow. The transformation works in a similar way as the transformation of OR-forks in the e^3 -value model. We begin from the start stimuli and

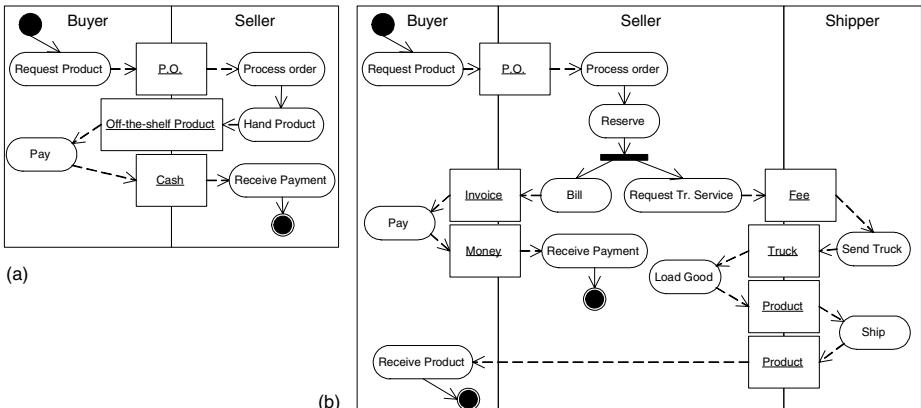


Fig. 6. Result of transformation step 1

we follow the execution flow. Each time we encounter a choice, we duplicate the model and substitute the choice with a direct transition to one of the alternatives. We cut the disconnected branches from the execution tree. This transformation step generates potentially many activity diagrams from a single activity diagram. Fig. 6 shows the result of transformation step 1 applied on the activity diagram from Fig. 2.

Step 2: Transformation tables. This step identifies the flow of messages between two swimlanes that result in an exchange of a value object in the e^3 -value-model sense. Additionally in each sequence, a single message is selected to be further transformed to a common object.

The selected messages and their sending and receiving swimlanes are mapped to common value objects and business units, respectively. The mappings are represented in transformation tables which for our example are Table 1 (b) and Table 2 (b).

Step 3: Generate reduced models. This step transforms each activity diagram representing an alternative into a reduced model. In particular, swimlanes and messages are transformed into business units and common value objects as specified in the mapping tables Table 1 (b) and Table 2 (b). As a result of this transformation the specific information on sequence of message exchanges is omitted. The final reduced models, derived form the activity diagram (see Fig. 2), are depicted in Fig. 7.

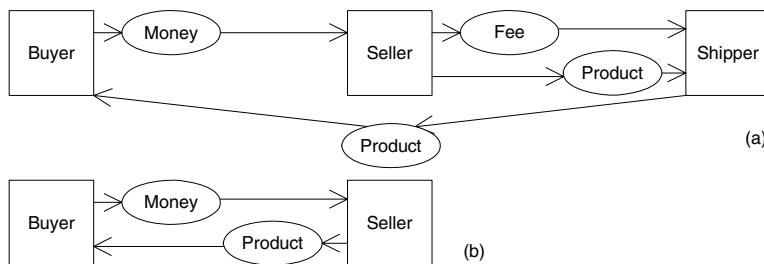


Fig. 7. Reduced models corresponding to the activity diagram

For clarity of presentation, we refer further to a reduced model derived from an activity diagram as a reduced process model. Respectively, a reduced model derived from a value model is referred to as a reduced value model. The origin of a reduced model is undistinguishable from the model itself; we name them differently for explanation purposes only.

3.5 Equivalent Reduced Models

Two reduced models are equivalent if each contains the same common value exchanges. This means that:

- each reduced model contains the same business units;
- each reduced model contains the same common value objects;
- in each reduced model, the sending and receiving business units of a particular common value object are the same.

In our example, we can determine that the reduced models of Fig. 5(b) and Fig. 7(b) are equivalent. In contrast, the models of Fig. 5(a) and Fig. 7(a) are not because (i) the Transport common object is not present in the reduced process model and (ii) the Product common object is exchanged between different units in the two models.

3.6 Transitivity

The reduced models in **Fig. 5(a)** and **Fig. 7(a)** can be made equivalent by applying transitivity on the Product common object in the reduced process model (see Fig. 7(a)). Transitivity removes intermediary units from a chain of common exchanges by directly representing the common exchange between the units in the beginning and the end of the chain. The reason for the unit in the beginning of the chain to involve additional units must be beneficial to the unit itself. Thus, the benefit must be provided by the intermediary unit because otherwise it would not have been involved.

As a consequence, the Product common exchanges between Seller and Shipper, and between Shipper and Buyer (see Fig. 7(a)) is represented as a direct Product common exchange between Seller and Buyer. The benefit introduced by the intermediary unit, i.e. Shipper, to Seller is represented by an unspecified common exchange. The unspecified common exchange can be instantiated by any reduced model common object. This introduces several additional options to be checked by the equivalence testing. With regard to the example, the instantiation with the Transport common object, as depicted in Fig. 8, results in equivalent reduced models.

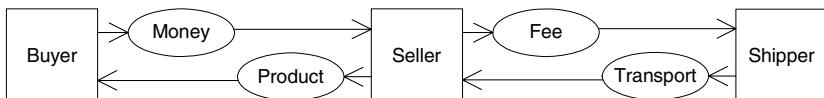


Fig. 8. Reduced model after transitivity transformation

Due to the equivalence of the reduced models, we consider the e^3 -value model and the activity diagram to be consistent, as we intuitively do in Section 2.3.

3.7 Consistency

An e^3 -value model and an activity diagram are consistent if there exists at least one non-trivial mapping under which the corresponding sets of reduced models are equivalent.

A non-trivial mapping is one for which holds that:

(1) every product value exchange in the e^3 -value model is mapped to one common value exchange. This includes that (i) every product value object is represented in the reduced model and (ii) sending and receiving actors of a product value object are not mapped to a single business unit in the reduced model;

(2) every transaction in the activity diagram is mapped to one common value exchange. This includes that (i) every transaction is represented in the reduced model

and (ii) sending and receiving swimlane of the message representing the transaction are not mapped to a single business unit in the reduced model.

The restrictions listed above preserve the product value exchanges in the e^3 -value model and the transactions in the activity diagram during the transformation; i.e., these are all represented in the reduced model. In case the granularity of the e^3 -value model and the activity diagram is similar, the relationships between actors and swimlanes, and between value objects and messages are usually one-to-one or one-to-many. Nevertheless, the relationships are many-to-many in the general case. The consequences of which, we discuss in the next section.

4 Granularity of Models: Many-to-Many Relationships Between Instances

In the previous section, we show how our consistency check works between models with comparable granularity. In particular, the relationships between actors and swimlanes, and value objects and messages are only one-to-one and one-to-many; where, the latter one breaks down to one-to-one relationships when the alternatives are taken separately. However, there is no guarantee that the granularity of e^3 -value models and activity diagrams is not the same. Therefore, the relationships between actors and swimlanes, and value objects and messages are in the general case many-to-many. In this section, we discuss when and, if so, how consistency of models with different granularity can be checked.

Our consistency definition is based on equivalence of reduced models, which requires equivalence of units and common objects. From (i) the equivalence of units, respectively common objects, and (ii) the way the mapping tables are constructed (see Section 3.2) follows that the semantic relationships between instances in the e^3 -value model and the activity diagram are one-to-one. To guarantee a proper result of our consistency check, we have to ensure that the checked models have similar granularity.

There are two strategies for adapting granularity: either aggregation is performed on the more fine-grained model or division of the more coarse-grained model. We have explored both approaches and discovered the following drawbacks.

The *aggregation approach* may lead to a single actor and a single swimlane. This is because, a many-to-many relationship between actors and swimlanes can result in an aggregation of two swimlanes which may trigger an aggregation of two actors and so forth. Due to the aggregation, the exchanges of product value objects between aggregated actors in the e^3 -value model are lost; the same holds for messages between swimlanes in the activity diagram. This loss of information makes the consistency decision less precise.

The *division approach*, on the other hand, may lead to the finest-grained granularity in both models, where a single actor exchanges a single value object or a single swimlane sends a single message. This is again a loss of information comparable to the aggregation case. In particular, the relation information between different value objects, respectively messages, is lost. Thus to limit the loss of information, we can constrain ourselves to division of only value objects and swimlanes or only actors and messages. From the two, we select the second because

actors are intuitively more coarse-grained than swimlanes and a single message may represent more than a single economic value. From a preliminary evaluation, this option is the most promising one, although, a more detailed evaluation has to be done.

Below, we illustrate how granularity of models is equalized by division of actors and messages. On behalf of an example, we show how we resolve one-to-many relationships.

4.1 Example

We consider a business case where a client is interested in the mortgage and insurance products of a bank. Fig. 9 represents the e^3 -value model of the example. The client (to the left in the figure) is interested in exchanging monthly fees for a period of time in case it gets a loan in a form of a mortgage. Additionally to that for some economic reasons, the client is interested in insurance from the same bank. The dependency path includes an AND-fork, shown within the Client actor, which denotes that the client wants both products.

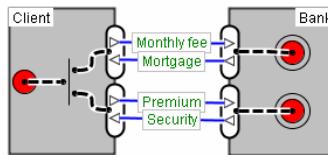


Fig. 9. Value model of the example

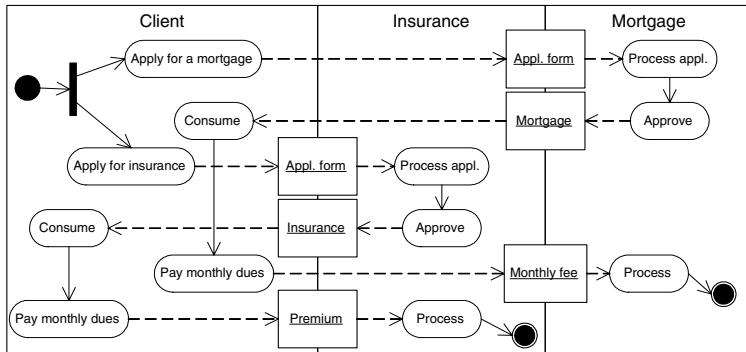


Fig. 10. Process model of the example with two swimlanes representing the bank

Fig. 10 represents an activity diagram for the above example case, where the bank is modeled as two swimlanes representing its Mortgage and Insurance departments separately. The client (the first swimlane from left to right) requests simultaneously a mortgage and insurance. The handling of the requests is performed in a similar way: e.g., the mortgage request (in the top of the figure) is processed by the Mortgage

swimlane which after processing the request grants mortgage to the client. Once given, the insurance and the mortgage are utilized by the client, which is shown in the client swimlane as a *Consume* activity. The bottom of Fig. 10 shows the monthly payments performed by the client: two separate payments to the Mortgage and Insurance departments are represented by the *Monthly fee* and *Premium* messages.

4.2 Splitting of Actors

The models in Fig. 9 and Fig. 10 differ in granularity: the bank from the e^3 -value model is represented as two individual swimlanes in the activity diagram. To resolve the one-to-two relationship between actor and swimlanes, we split the bank actor in the e^3 -value model. The newly appeared actors, named *BankM* and *BankI*, need to distribute the value exchanges of the original actor *Bank*. In our example case, there are four exchanges and we generate all possible combination with the new actors. We check consistency with all combinations.

The choice of splitting the *Bank* actor is derived from the mapping tables, where we observe the one-to-many relationship. We split an actor by splitting the corresponding unit in the reduced model.

Based on the splitting of the units in the value reduced model it turns out that the reduced models are equivalent, which fits to the intuitive consistency.

4.3 Splitting of Messages

Fig. 11 shows a second activity diagram for the example case. The diagram differs in two points from Fig. 10. First, the bank is represented as one swimlane and correspondingly the activities and messages belonging to the Mortgage and Insurance swimlane are in the Bank swimlane. The second difference is in the way payment of monthly dues is modeled. In the bottom of Fig. 11, payment by the client is represented as a single message.

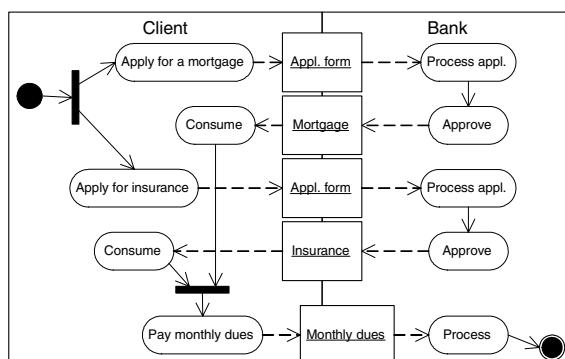


Fig. 11. Process model of the example with one message representing the monthly dues

The models in Fig. 9 and Fig. 11 differ in granularity: the monthly payments are represented as two distinctive value objects in the e^3 -value model while in the activity diagram they are modeled as one message. To resolve the two-to-one relationship between value objects and a message, we split the `Monthly dues` message into `IMonthly dues` and `MMonthly dues`. The new messages share the same sender and receiver as the original message.

The choice of splitting the `Monthly dues` message is derived from the mapping tables, where we observe the one-to-many relationship. We split a message by splitting the corresponding common object in the reduced model.

Based on the splitting of the common objects in the process reduced model it turns out that the reduced models are equivalent, which fits to the intuitive consistency.

4.4 Implications

We have analyzed a number of examples to justify our approach of splitting actors and messages. We classify them based on:

- Cardinality of the relationship, where we consider one-to-many and many-to-many relationships;
- Direction of the relationship, where we consider one instance in the e^3 -value model related to many instances in the activity diagram and vice versa;
- Type of the relationship, where we specialize relationships into *individual* and *aggregation*. An individual one-to-many relationship means that one entity is mapped to several independent entities each of which represents the entity as a whole. (This is the type of relationship we have in our first example in Section 3.) An aggregation one-to-many relationship means that one entity is mapped to several independent entities which together represent the entity as a whole. (This is type of relationship we have in our second example in Section 4);
- Arguments of the relationship, where we considered relationships between actors and swimlanes, and value objects and messages.

Our analysis of all combinations of the classifications criteria above shows that it is possible to adapt the granularity of models applying the approach of splitting actors and messages. It is possible to reduce the one-to-many and many-to-many relationships to one-to-one relationships in all cases except one where we have one-to-many aggregation type relationship between a swimlane and actors. Although such a relationship is possible, we think it is rarely used; intuitively, an e^3 -value model is at a higher level of granularity than an activity diagram.

5 Validation

The proposed consistency check is valid with respect to the intuitive consistency definition if all model pairs considered to be intuitively consistent are consistent with regard to our consistency definition and vice versa. To argue that this is the case, we will decompose the intuitive consistency definition from Section 2.3 and compare it with the building blocks of our consistency definition.

For the intuitive consistency, we make the following observations:

1. It is based on relations between separate alternative dependency paths and separate execution sequences;
2. The relation between an alternative dependency path and an execution sequence is based on a single set of product value exchanges happening in both models.

Our transformation procedures represent the original model as several reduced models, one per alternative, which is based on alternative dependency paths and execution sequences. That is, one alternative dependency path (execution sequence) results in a single reduced model. Thus, the granularity of the performed consistency check is the same as in the intuitive one.

The second observation says that an alternative dependency path and an execution sequence result in the same product value exchanges. Our definition of equivalent reduced models requires identical common value exchanges in the two models. This shows that both consistency definitions require a relationship between models based on *the same set of* product value exchanges and *on the same set of* common value exchanges.

As we describe in Section 3 the relationship between value exchanges and common exchanges is one where every product value exchange is represented in the reduced model. Similarly, transactions in the activity diagram are identified as such if they result in a product value exchange. Thus in case of a non-trivial mapping, every product exchange is transformed to a common exchange.

We conclude that the proposed consistency definition is valid with respect to the intuitive consistency definition.

6 Consistency with the Application Communication Perspective

Throughout this paper, we discuss the economic value and business processes perspectives and their consistency relationship (see Section 1). The two perspectives focus on the value and control flow aspects among businesses. Our third perspective, i.e. the application communication perspective, models the data flow without explicating alternatives in a similar sense to alternatives in the value and process models. The differentiation between data-flow alternatives is based on knowledge gained from the development process of the communication model. Thus, each alternative in the application communication perspective is modeled separately, represented as a partial communication model.

From a preliminary investigation, we can state that the consistency check approach based on alternatives can be applied on the remaining two consistency relationships between a value model and a communication model, and between a process model and a communication model. However, a more detailed analysis will be performed in future work.

We select the consistency relationship between the e^3 -value model and the activity diagram as we consider it the most difficult one. An e^3 -value model includes several alternatives, which represent several possible scenarios of value exchanges. Respectively, an activity diagram includes choices which result in several possible execution sequences. In comparison with the application communication perspective,

the additional steps to separate the alternatives in an e^3 -*value* model and an activity diagram make the consistency checking more complex.

7 Related Work

Consistency can be checked in various ways. The approach with *syntactic translation* (also called direct translation) [2] is based on directly relating terms of two notations. Then, one specification is translated to the modeling language of the other. The *common semantic model* (also called canonical representation) approach [3] selects a single modeling notation (not necessarily one already in use) and transform all models to that notation. The *meta-representation* approach [11, 12] does not require transformation between models. It specifies relations between meta-modeling and modeling concepts from each modeling notation. These relations must hold between the concepts and their instances in each model.

Our approach is based on the common semantic model approach. We define a common semantic model, which we call reduced model, in a pair-wise fashion. This gives us richer reduced models compared to a single reduced model for all perspectives. Additionally, our approach introduces a consistency check based on alternatives; i.e. models are decomposed into smaller models and checked individually for consistency. This provides a consistency check that matches with the intuitive consistency definition.

Our work is an extension of Gordijn's [8] requirement engineering approach to innovative e-commerce ideas. He specifies a method for exploration of business opportunities based on the distribution of value in business networks. Additionally to the value viewpoint (viewpoint is a synonym for perspective), the approach includes two more: a business process and an information systems viewpoints. The three viewpoints match closely with our perspectives. However, we explicitly check for consistency as we assume independent development of models; whereas Gordijn's approach is based on a common set of scenarios represented in each model.

Our approach requires a semantic mapping between concepts in the value and process models. The work of Gordijn, Akkermans and Vliet [6] elaborates on the differences between business and process modeling by showing semantic differences between concepts. We use this information to define our transformation tables. While Gordijn, Akkermans, and Vliet specify differences between concepts of the different models. The proposed approach specifies semantic relationships between instances.

Wieringa and Gordijn [13] define a correctness relationship between an e^3 -*value* model and a process model. We use this to define our intuitive consistency. In addition, we provide an operationalization of this intuitive consistency definition based on transformations to reduced models.

The work of Dijkman et al. [4] is also based on the common semantic model approach. It relates viewpoints (viewpoint is a synonym for perspective) by means of a basic viewpoint which contains pre-defined concepts and relations. Every viewpoint from a design framework need to be mapped to basic concepts and relations from the basic viewpoint. Our approach differs in the way how the reduced model (the basic viewpoint in Dijkman et al.'s terms) is defined. We do not require a pre-defined reduced model with abstract basic constructs, but we determine the reduced model

after the modeling notations are selected. As we pointed in Section 0, this allows defining richer reduced models in terms of common concepts and relations.

Consistency of a workflow model can usually be defined based on the set of potential execution sequences, a straight forward approach to check consistency is on a single workflow model. This approach has been applied to several workflow models, like for example by v.d.Aalst and Weske [1] to Workflow Nets (WF-Nets), by Fu et.al. to guarded Finite State Automata [5], by Yi and Kochut [17] to Coloured Place/Transition Nets, or by Wodtke and Weikum [15] to statecharts. In either case it is checked whether the execution of the workflow results in a deadlock, that is, no further action is possible although a final state has not been reached yet. However, there exists also approaches on checking consistency between several workflows represented in the same modeling approach, like e.g. [1, 16, 9]. In our paper, consistency between different modeling approaches is defined.

8 Conclusions and Future Work

The contribution of this paper is a definition of consistency between an e^3 -value model and an activity diagram. We operationalize the consistency check by defining a reduced model that contains the common concepts from two models. Further with the help of mapping tables, we transform the e^3 -value model and the activity diagram to reduced models. Finally, we check equivalence of reduced models. We argue that the consistency definition is valid with respect to the intuitive consistency definition.

The e^3 -value model and the activity diagram represent two of the three perspectives of a development framework used to align information system capabilities and business objectives [18]. The third perspective represents the communication among components of the business information systems. Future work includes the definition and operationalization of the remaining two consistency relationships: between the e^3 -value model and the communication perspective model, and between the activity diagram and the communication perspective model. Further, we aim to investigate the usefulness of the three binary consistency definitions as necessary conditions for a global consistency. Finally, an implementation has to be provided to automate the consistency check of the different perspectives as more realistic and complex examples are likely to contain high numbers of reduced models.

Acknowledgements

We thank Roel Wieringa and Maya Daneva for their valuable comments on the draft versions of this paper.

References

1. Aalst van der, W.M.P., Weske, M.: The P2P approach to Interorganizational Workflows. Proceedings of 13th International Conference on Advanced Information Systems Engeneering (CAiSE). Interlaken, Switzerland (2001)
2. Bowman, H., Steen, M.W.A., Boiten, E.A., Derrick, J.: A formal framework for viewpoint consistency. Formal Methods in System Design, 21, September (2002) 111—166

3. Braatz, B., Klein, M., Schröter, G.: Semantical Integration of Object-Oriented Viewpoint Specification Techniques. In *Integration of Software Specification Techniques for Applications in Engineering*, Lecture Notes in Computer Science, Springer. (2004)
4. Dijkman, R.M., Quartel, D.A.C., Pires, L.F., Sinderen van M.J.: An Approach to Relate Viewpoints and Modeling Languages. In: *Proceedings of the 7th IEEE Enterprise Distributed Object Computing (EDOC) Conference*, Brisbane, Australia (2003) 14—27
5. Fu, X., Bultan, T., Su, J.: Realizability of Conversation Protocols with Message Contents. *Proceedings IEEE International Conference on Web Services (ICWS)* (2004) 96—103
6. Gordijn, J., Akkermans, J.M., Vliet van, J.C.: Business Modelling is not Process Modelling. In: *Conceptual Modeling for E-Business and the Web*, LNCS 1921. Salt Lake City, USA, October 9-12 (2000) 40—51
7. Gordijn, J., Akkermans, J.M.: Value based requirements engineering: exploring innovative e-commerce idea. *Requirements Engineering Journal*, 8, 2 (2003) 114—134.
8. Gordijn, J.: Value based requirements engineering: Exploring innovative e-commerce ideas. PhD. thesis, Vrije Universiteit Amsterdam (2002)
9. Kindler, E., Martens, A., Reisig, W.: Inter-operability of Workflow Applications: Local Criteria for Global Soundness. *Business Process Management, Models, Techniques, and Empirical Studies* (2000) 235—253
10. OMG: OMG UML Specification (2003) <http://www.omg.org/cgi-bin/doc?formal/03-03-01>
11. Sunetnanta, T., Finkelsteing, A.: Automated Consistency Checking for Multiperspective Software Specifications. *Workshop on Advanced Separation of Concerns*. Toronto (2001)
12. Sunetnanta, T.: Multiperspective Development Environment for Configurable Distributed Applications. Ph.D. Thesis. Department of Computing, Imperial College. February (1999)
13. Wieringa, R.J., Gordijn, J.: Value-Oriented Design of Service Coordination Processes: Correctness and Trust. *ACM Symposium on Applied Computing* (2005)
14. Wieringa, R.J.: Design Methods for Reactive Systems. Morgan Kaufmann (2002)
15. Wodtke, D., Weikum, G.: A Formal Foundation for Distributed Workflow Execution Based on State Charts. In: Afrati, F.N., Kolaitis, P. (eds.): *Proceedings of the 6th International Conference on Database Theory (ICDT)* (1997) 230—246
16. Wombacher, A., Fankhauser, P., Aberer, K.: Overview on Decentralized Establishment of Consistent Multi-Lateral Collaborations Based on Asynchronous Communication. *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE)* (2005) 164—170
17. Yi, X., Kochut, K.J.: Process Composition of Web Services with Complex Conversation Protocols: a Colored Petri Nets Based Approach. *Proceedings of the Design, Analysis, and Simulation of Distributed Systems* (2004) 141—148
18. Zlatev, Z., Daneva, M., Wieringa, R.J.: Multi-Perspective Requirements Engineering for Networked Business Systems: A Framework for Pattern Composition. *8th Workshop on Requirements Engineering*. Porto, 13-14 June (2005)

Maintaining Global Integrity in Federated Relational Databases Using Interactive Component Systems

Christopher Popfinger and Stefan Conrad

Institute of Computer Science,

University of Düsseldorf,

D-40225 Düsseldorf, Germany

{popfinger, conrad}@cs.uni-duesseldorf.de

Abstract. The maintenance of global integrity constraints in database federations is still a challenge since traditional integrity constraint management techniques cannot be applied to such a distributed management of data. In this paper we present a concept of global integrity maintenance by migrating the concepts of active database systems to a collection of interoperable relational databases. We introduce *Active Component Systems* which are able to interact with each other using direct connections established from within their database management systems. Global integrity constraints are decomposed into sets of *partial integrity constraints*, which are enforced directly by the affected Active Component Systems without the need of a global component.

1 Introduction

Many organizations or companies own a multitude of data sources, which generally emerged autonomously to fit the needs of a department or a certain group of users at a local site. Local applications produce or modify data that is often semantically related to data stored on a different autonomous source. One of the main challenges in the integration of data in such environments is the autonomy of the data sources. This autonomy implies the ability to choose its own database design and operational behavior. Local autonomy is tightly attached to the data ownership, i.e. who is responsible for the correctness, availability, and consistency of the shared data. Centralizing data means to limit local autonomy and revoke the responsibility from the local administrator, which is not reasonable in many cases. A federated architecture for decentralizing data has to balance both, the highest possible local autonomy and a reasonable degree of information sharing [1]. Hence, the architecture of a company wide information system has to be applicable to the data policy of the company and vice versa. To be more precise, the question of data ownership determines the composition of the company wide information platform, while it has to ensure a high level of consistency and fail-safety.

In many scenarios, local data is, and should be, exclusively manipulated by local applications, whereas global applications are used to display and analyze this data without the need for global write transactions. For example, a company could store employee data in a database of the management department, whereas the research department maintains information about ongoing projects and researchers in its own database. Each department is responsible for the up-to-dateness and correctness of its data. Information about employees and research projects shall be integrated and displayed on the company's website. Obviously the data in both autonomous databases is interrelated, since all the researchers are employees of the company. Thus, we need to ensure that every entry in the research database has a corresponding entry in the management database. Interdependencies of data stored on multiple databases can be considered as integrity constraints expressed over a global schema that is derived from relevant schemata of the local databases. The maintenance of these global integrity constraints is still a problem since traditional integrity constraint management techniques cannot be applied to such a distributed management of data. Our work is developed in the context of relational databases, since this type of data source is widely used for data storage in practice. We assume an information system which comprises a collection of autonomous relational sources of various vendors running on different platforms. The databases store interdependent data that is accessed by local and global applications.

In this paper we present a concept for global integrity maintenance in such a federated relational database systems by extending the concepts of active database systems to a collection of interoperable relational databases. We introduce *Active Component Database Systems* (ACDBS), which are able to interact with each other using direct database connections established from within their database management systems. Interdependencies between the data sources are expressed as global integrity constraints and enforced using constraint checks that are entirely implemented on the ACDBSs without the need of a global component or federation layer. At the same time, we allow the component database systems to retain the greatest possible extent of local autonomy.

The remainder of the paper is organized as follows. We start with an introduction to Active Component Database Systems as the main components of our architecture in section 2. Section 3 defines partial integrity constraints as a new type of constraints suitable for ACDBS, followed by a detailed explanation of global integrity checking based on partial integrity constraints in section 4. Section 5 discusses properties of our concepts, while related work is presented in section 6. Section 7 concludes and draws up future work.

2 Active Component Database Systems

We start with an introduction to Active Component Database Systems (ACDBS) as the main concepts of our approach. An ACDBS is an autonomous component database system or component system (CDBS or CS) of a federated database system as described in [2]. The active functionality of this kind of component

systems, which we are going to describe in the following, can be used to ensure consistency and to enforce business rules in both, tightly coupled and loosely coupled federations. Within the classical notion of federated databases, the component systems do only have passive functionality regarding the federation. Like repositories, they provide access to their data and respond to data requests initiated by the clients. Such passive component database systems, with respect to the federation, operate isolated and do not have any knowledge of other CDBSs within the federation to which their data is related.

Active database systems, which are not automatically ACDBSs when participating in a federation, assist applications by migrating reactive behavior from the application to the DBMS. They are able to observe special requirements of applications and react in a convenient way if necessary to preserve data consistency and integrity. The integration of active behavior in relational database systems is not particularly new and most commercial database systems support ECA rules, whereas the execution of triggers is mainly activated by operations on the structure of the database (e.g. insert or update a tuple) than by user-defined operations [3]. Unfortunately, the ability to check constraints in active databases, especially the scope of trigger conditions and actions, has until recently been limited to the isolated databases they were defined at. Subsequent developments integrated special purpose programming languages (e.g. PL/SQL [4]) into the database management system to overcome some limitations of the query language and to provide a more complex programming solution for critical applications. But again, the scope of these extensions was strictly limited to the system borders of the database system, so an interaction with its environment was impossible. Thus, the support of ECA rules and triggers is necessary, but not sufficient for the concept we propose here.

Latest developments, especially in commercial database systems, take the functionality of active databases beyond former limits. The significant improvement, on which this work is based on, is the ability of modern active database systems to execute programs written in a standalone programming language from within triggers, user defined functions, or stored procedures.

Definition 1. *The ability of a database system to execute programs or methods from within its DBMS to interact with software or hardware components beyond its system border shall be called enhanced activity. A database with enhanced activity is an Enhanced Active Database System (EADBS). The execution of a program or method in this context shall be called an External Program Call (EPC).*

The execution of external programs (EPs) from inside the DBMS offers new perspectives to data management and processing in an information sharing environment. Besides the maintenance of global integrity constraints as presented here, it can be used to improve communication with other external components like database wrappers [5]. In this paper, we use the database connectivity of the programming language to add the following functionalities to a component database system of a federation:

Query the state of a remote database: The main functionality which is elementary for our approach is the ability of an CDBS to query a remote data source *directly* during the execution of a database trigger. After a connection has been established by the EP, we can perform any read operation on the remote schema items we are allowed to access. Depending on the query language we can formulate complex queries with group and aggregate functions (e.g. like in SQL). The query result of the remote database is used locally to evaluate conditions of ECA rules. We call this kind of query a *remote state query*.

Manipulating a remote database: After a connection is set up by the program, a CDBS is basically able to modify the data stock of the remote database *directly* during the execution of a database trigger. Assuming the appropriate permissions, any operation supported by the query language can be executed including data insertions, updates, and deletions. Depending on the query language, a CDBS is thus basically able to modify even the schema of a remote database using for example `ALTER TABLE` statements in SQL. In the following, a manipulation of remote data or schema items from within a database trigger shall be called an *injected transaction*, since its execution depends on a triggering transaction on a local relation.

The programming language used for EPs has to provide the functionality to open and close a connection to a remote data source and execute queries upon that data stock. Furthermore, we must be able to pass parameters to the EP and to access the corresponding program output from inside the trigger. This output can be used to evaluate trigger conditions or to determine subsequent trigger actions. Since the EPs are embedded straight inside the DBMS of the local system, we are able to delay or abort transactions depending on the state of another data source or the result of an injected transaction. Just like common triggers that exclusively use local data to evaluate their trigger conditions, the DBMS autonomously schedules the execution of the trigger that encapsulates the EP. In particular, we do not force a component system to provide an atomic commitment protocol like 2PC. From the point of view of the remote database, a query of another DBMS via the database server is handled like a request of an ordinary application.

Within recent commercial database systems a commonly supported programming language which meets the requirements just mentioned is Java (e.g. Java Stored Procedures or Java UDFs [6, 4]). It contains JDBC, a common database connectivity framework, to provide a standardized interface for a multitude of different data sources like relational databases or even flat files. During the execution of an EP based on Java, we use JDBC to connect to remote data sources from within triggers. After a connection has been established, we execute queries using SQL as a standardized query language. Our concept can be adapted to other relational database systems supporting different programming languages that fulfill the requirements listed above.

Definition 2. *An Active Component Database System (ACDBS) is an EADBS, which actively participates in maintaining global integrity constraints in*

a federation. It is able to directly communicate with other component systems, to which its data is semantically related, and implements constraint checks to maintain consistency among this interdependent data.

Constraint checks performed by ACDBSs are entirely implemented and executed on local CDBSs, but require access to remote data. Since these checks cannot be expressed by either local or global constraints, we introduce *partial integrity constraints* as a new type of integrity constraints for ACDBS.

3 Partial Integrity Constraints

In this section, we discuss partial integrity constraints as the basic concept of our approach. As already mentioned, we assume a federation of relational databases. Each ACDBS in this federation has to meet two requirements concerning the programming language for encoding EPs: (1) It must be able to connect to other component systems of the federation using the database connectivity of the programming language and (2) it must support a query language understood by the other component systems to execute at least read operations on the remote data stock. In practice, two widely used standard database connectivity interfaces are JDBC and ODBC, which support a multitude of relational databases. An established query language for relational databases certainly is SQL. This enhanced functionality is used to implement constraint checking algorithms for partial integrity constraints, which are defined next.

3.1 Definition of Partial Integrity Constraints

We start with the following definitions similar to [7]:

Definition 3. A federation F of relational component systems is a set of n interconnected database systems $\{S_1, \dots, S_n\}$. The database systems do not necessarily have to be located on physically different nodes of the network. Each system $S_i \in F$ manages a local database D_i . A local schema \mathcal{D}_i of a database D_i comprises the schemata $\mathcal{R}_1^i, \dots, \mathcal{R}_{n_i}^i$ of the relations $R_1^i, \dots, R_{n_i}^i$ stored in the database. The global database schema \mathcal{G} of F is the set of all relational schemata \mathcal{R}_j^i in F .

We assume that a real-world object, that is modeled in a component database of F , is globally identified by a set of key attributes, i.e. a real-world object will have the same key attribute values when stored in different CDBSs. Otherwise we assume mapping functions to match real-world objects in different sources.

Definition 4. A local integrity constraint $I_L^{D_i}$ is a boolean function over a local database schema \mathcal{D}_i , i.e. $I_L^{D_i} : \mathcal{D}_i \rightarrow \{\text{true}, \text{false}\}$. A global integrity constraint I_G is a boolean function over the global schema \mathcal{G} , i.e. $I_G : \mathcal{G} \rightarrow \{\text{true}, \text{false}\}$. It cannot be expressed over a local database schema $\mathcal{D}_i \in \mathcal{G}$. Constraint checks for $I_L^{D_i}$ and I_G are algorithms for evaluating $I_L^{D_i}$ and I_G respectively.

After a global constraint I_G has been defined over \mathcal{G} , we identify a non-empty set $\mathcal{C} \subseteq F$ of component databases $c \in \mathcal{C}$ whose local schemata \mathcal{R}_c are affected by I_G , i.e. data stored in the relations R_c on the component databases is semantically related. Thus, from the point of view of each component database c , I_G affects a relation managed locally and at least one remote relation managed by another component system. For example, if a key constraint is defined on a global attribute that is derived from multiple sources, each of the sources has to ensure the global uniqueness of the key attribute, when a new tuple is inserted locally. This means that I_G consists of a set of *partial integrity constraints*, which we define as follows:

Definition 5. A partial integrity constraint I_{R_c} on an ACDBS $c \in \mathcal{C}$ is a boolean function, which is expressed over the local schema \mathcal{R}_c and related schemata \mathcal{R}_{k_u} for $k_u \in \mathcal{C} \setminus \{c\}$, i.e. $I_{R_c} : \mathcal{R}_c \times \mathcal{R}_{k_1} \times \dots \times \mathcal{R}_{k_v} \rightarrow \{\text{true}, \text{false}\}$ for $v \leq |\mathcal{C}| - 1$. A constraint check for I_{R_c} is an algorithm for evaluating I_{R_c} , which is entirely implemented on c using external program calls to access the remote schemata \mathcal{R}_{k_u} .

A partial integrity constraint consists of a local constraint check and one or more remote constraint checks on interrelated remote data, depending on the type of global constraint and the number of affected databases. It is used to express a global constraint from the local view of a single component database. An ACDBS, which implements a partial integrity constraint, has to ensure consistency of its local data depending on related data stored in other component databases. This means that it is responsible for checking a specific part of the corresponding global integrity constraint concerning local write operations on the interrelated data. Thus, a global integrity constraint is assured, iff all affected component systems enforce their corresponding partial integrity constraints, expressed as

$$I_G : \bigwedge_{c \in \mathcal{C}} I_{R_c}$$

The global constraint I_G consists of a conjunction of partial integrity constraints I_{R_c} , which are formulated as

$$I_{R_c} : \text{local}_{R_c} \wedge \bigwedge_{j \in \mathcal{C}'} \text{remote}_{R_c, R_j}$$

Depending on the type of global integrity constraint, each affected ACDBS c has to check an optional local condition local_{R_c} and one or more remote conditions. remote_{R_c, R_j} defines a pairwise dependency between the affected local relation R_c and one interrelated relation R_j on component j . Remote conditions do not necessarily have to be defined for each pair of local and remote databases in \mathcal{C} . Thus, $\mathcal{C}' \subseteq \mathcal{C} \setminus \{c\}$ denotes a subset of ACDBSs, which are required to check for a specific integrity constraint. A constraint check for I_{R_c} implements tests for the local condition local_{R_c} and each remote condition remote_{R_c, R_j} . For aggregate constraints, a test for a local or remote condition results in the successful computation of a local or remote aggregate. Detailed examples for partial constraint checks are provided in section 4.

3.2 Specification of Partial Integrity Constraints as ECA Rules

Since our concept of global integrity maintenance is based on ACDBS with enhanced activity, we use database rules following the event-condition-action (ECA) paradigm to specify a partial integrity constraint I_{R_c} on a component database c as follows:

```
define rule PartialIntegrityRule  $I_{R_c}$ 
on event which modifies  $R_c$ 
if a test for  $local_{R_c}$  yields false or
    a test for  $remote_{R_c, R_j}$  yields false
do local and/or remote action(s) to ensure or
    restore a consistent global state
```

Such integrity rules can precisely define both: events that potentially violate the integrity of local and remote data, and corresponding reactions on these events to ensure or restore consistency in the entire system. The relevant events concerning data consistency are modifications of the data stock, i.e. insertions, updates, and deletions. According to the definition of partial constraints, each rule condition and rule action of a partial integrity rule can consist of a local and one or more remote checks. The local check $local_{R_c}$ exclusively uses and accesses local data, while the remote checks $remote_{R_c, R_j}$ exclusively process data stored on remote systems. The remote checks of a partial integrity rule are implemented using remote state queries (or injected transactions) provided by the ACDBS. Thus, we have the following options to call an external program during an integrity check:

During the evaluation of a trigger condition: An external program call during the evaluation of a trigger condition allows a DBMS to determine subsequent actions depending on the result of a remote state query or the result of an injected transaction. Thus, a locally executed constraint check can be conditioned by the state and behavior of a remote data source. In our concept, most of the constraint checks are implemented using remote state queries from within trigger conditions. The part of the trigger condition, which evaluates a condition using remote data shall be called *remote condition*.

During the execution of (a) trigger action(s): Besides the remote condition, an external program can be executed as a trigger action. A local transaction can thus trigger an injected transaction to manipulate a remote data source. This can be used to execute consistency restoration actions or to implement special constraints like cascading referential integrity. This part of the trigger action, which manipulates remote data using injected transaction shall be called *remote action*.

The specific combination of the time the corresponding EP is executed (i.e. during remote condition or remote action) and the time a partial integrity rule is evaluated (i.e. **before** or **after** a local transaction is committed) is significantly affecting the behavior of the entire system. Please note, that we are certainly not limited to exclusively one of these combinations. During the evaluation of a partial integrity

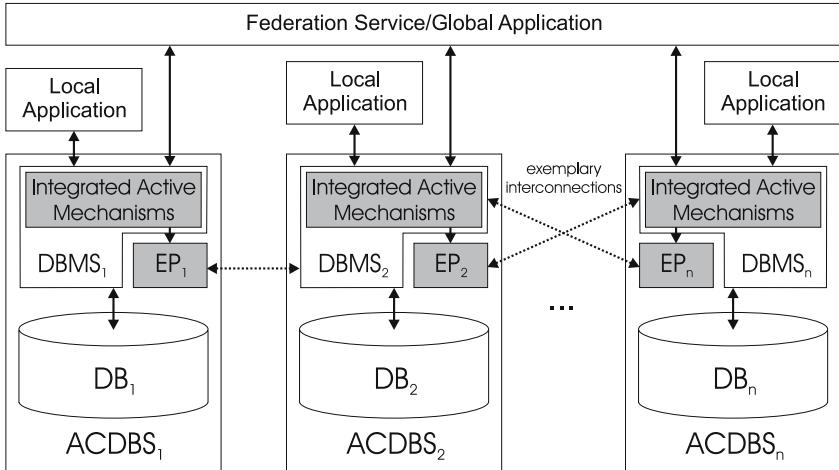


Fig. 1. Basic Architecture

rule, external programs can be called from both, the remote condition and the remote action, before or after a local transaction is committed.

Putting it all together, we present the architecture for global integrity checking in heterogeneous information systems using active component systems depicted in Fig. 1. The partial integrity checks are defined and implemented directly in the ACDBSs, building up an application independent communication layer to jointly ensure global consistency of interdependent data. The ACDBSs call EPs to check remote conditions or to execute remote actions of locally defined partial integrity constraints. Each transaction is checked according to local and partial integrity constraints, no matter if submitted by local or global applications. The maintenance of global integrity is thus migrated from a global application or federation layer to the underlaying active component systems.

3.3 System Interaction

We now give a schematic description of the interaction process between two ACDBSs during the execution of a partial integrity check. Consider two relations R and S on active component systems $ACDBS_1$ and $ACDBS_2$, which store interdependent data. To enforce a global constraint, we have to define and implement partial constraint checks on both component systems. Therefore we create the following objects on each ACDBS (see Fig. 2):

- An **external program (EP)** (here a Java method) to execute queries upon the remote data stock,
- a **user defined function (UDF)**, which is mapped to the external Java method, and
- a **trigger** which executes the UDF when relevant write operations occur on the relation.

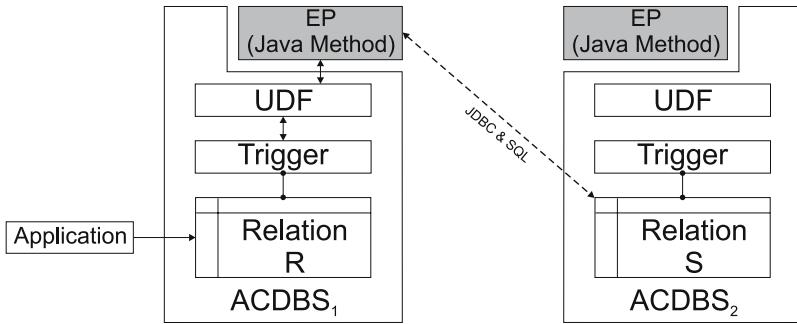


Fig. 2. Interaction between two Active Component Systems

We assume a global key constraint to be enforced on R and S , i.e. whenever a tuple is inserted or modified in R or S , we have to check the global uniqueness of the key attributes of the newly inserted or modified tuple in both relations. Our description focuses on data modifications in R , since modifications in S would be processed analogously. When an application inserts or updates data items ΔR in R , the corresponding trigger is executed by the database system *before* the transaction is completed. The trigger has access to ΔR via temporary tables provided by the DBMS. For each new tuple r in ΔR , the trigger first performs a check on the local data and afterwards, if necessary, on the remote data. If the local test fails, the key constraint is already violated and the remote test is omitted. If the local test succeeds, the trigger calls a UDF to check for conflicts in the remote data. The UDF is mapped to a Java function and receives r as a parameter from the trigger.

The Java function now bridges the gap between the two component systems. Using JDBC it connects to the remote database and executes an SQL query to check for the existence of r in S . The function returns *true* or *false* depending on the query result. The trigger receives this result and is now able to determine subsequent actions. Please keep in mind that, for the scope of this paper, we execute local and remote checks during the evaluation of the trigger condition before the transaction is completed. Thus, the transaction is blocked as long as the trigger is executed. If a corresponding tuple already exists in S , then we reject the data modifications on R . An SQL error is raised to signal the global key constraint violation.

4 Checking Global Integrity Constraints

We now concretize our concept of global integrity maintenance explaining how partial integrity constraints are expressed and implemented for different constraint types. Therefore we use a simplified scenario with two relational data sources. A generalization to more than two sites is discussed in section 5.

Consider a company with two research departments A and B , which both manage their own autonomous relational database DB_A and DB_B . The company wants to integrate these standalone sources into an information system and define

Table 1. Example relations in the research departments A and B

Department	Database	Relations
A	DB_A	$resA(R_A, N_A, S_A)$ $projA(P_A, T_A, B_A)$ $proresA(R_A, P_A)$
B	DB_B	$resB(R_B, N_B, S_B)$ $projB(P_B, T_B, B_B)$ $proresB(R_B, P_B)$

integrity constraints to ensure global data consistency within the entire company. We assume that the sources are relational databases with enhanced activity, which host the relations shown in Table 1. Each department stores information about researchers in relation res (researcher number R , name N , salary S) and their corresponding projects in relation $proj$ (project number P , title T , budget B). Researchers are related to projects using the $prores$ relation (m:n).

According to the classification presented in [8, 9], we consider four commonly used classes of global integrity constraints which can be defined on the global schema: attribute constraints, key constraints, referential integrity constraints, and aggregate constraints. In the following, we provide an example for each non-trivial class of global integrity constraints, followed by a rule definition for corresponding partial integrity constraint on the affected ACDBSs.

4.1 Attribute Constraints

The company could define a constraint saying that the budget of each project may not exceed a certain value. Since this global attribute constraint is expressed over a single attribute, it can be translated into a local attribute constraint and thus be enforced by local integrity mechanisms on DB_A and DB_B , e.g. by an additional **check** clause in both project relations. The global constraint can be enforced by a local constraint check on each ACDBS, so no EPCs are required.

4.2 Key Constraints

The company may want to ensure that each project is globally identified by a unique identifier, i.e. the values stored in P_A and P_B are globally unique. Thus, each time a project is added in one of the research departments, we have to ensure that the new project number does not already exist locally and in the project database of the other research department.

Since partial constraint checks are entirely implemented on a participating ACDBS using EPCs to access remote data, we decompose Key_G into a set of partial integrity constraints Key_{projA} and Key_{projB} for DB_A and DB_B respectively:

$$Key_G : Key_{projA} \wedge Key_{projB}$$

The partial constraints are in turn formulated as

$$\begin{aligned} Key_{projA} &: local_{projA} \wedge remote_{projA,projB} \\ Key_{projB} &: local_{projB} \wedge remote_{projA,projB} \end{aligned}$$

A partial constraint consist of a local condition and a remote condition, which are defined for Key_{projA} as follows (Key_{projB} is defined analogously):

$$\begin{aligned} local_{projA} &: \forall P, T, B, P', T', B' : \\ &\quad [projA(P, T, B) \wedge projA(P', T', B') \Rightarrow \neg(P = P')] \\ remote_{projA,projB} &: \forall P, T, B, P', T', B' : \\ &\quad [projA(P, T, B) \wedge projB(P', T', B') \Rightarrow \neg(P = P')] \end{aligned}$$

Suppose the tuple $projA(p, t, b)$ is inserted. According to Key_{projA} we have to check the existence of the key locally and in $projB$, stored on DB_B , with the following tests for $local_{projA}$ and $remote_{projA,projB}$:

$$\begin{aligned} localtest_{Key_{projA}} &: \exists P, T, B : [projA(P, T, B) \wedge (P = p)] \\ remotetest_{Key_{projA}} &: \exists P, T, B : [projB(P, T, B) \wedge (P = p)] \end{aligned}$$

Both tests are evaluated by performing queries on the relevant relations for a tuple that has p as its project number. Therefore, we need two boolean functions: $checklocalkey : schema(projA) \rightarrow \{\text{true}, \text{false}\}$ for $localtest_{Key_{projA}}$ and $checkremotekey : schema(projB) \rightarrow \{\text{true}, \text{false}\}$ for $remotetest_{Key_{projA}}$. $checklocalkey$ should always be evaluated first to avoid cost-intensive remote data access where possible. Please note that although the uniqueness of key attributes may already be enforced by an additional local key constraint, we need the local check in the partial constraint since in general it is not possible to access the result from a local constraint check from within an active component like a trigger.

The $checkremotekey$ function is implemented using a remote state query, which queries database DB_B to find tuples with the values to be inserted. If the query result is not empty or the remote source is not reachable by the external program, then the function is evaluated to $false$, i.e. the corresponding transaction in database DB_A is rejected. The condition is evaluated *before* the triggering operation is committed at DB_A . The corresponding partial rule for Key_{projA} is expressed as:

```
define rule PartialKeyConstraint
on creation of a new object in projA
if checklocalkey yields false or checkremotekey yields false
do reject transaction
```

A partial constraint is herewith realized on an ACDBS with an implementation of the ECA rule using two functions $checklocalkey$ and $checkremotekey$ with one remote state query. Having implemented both partial constraints Key_{projA} and Key_{projB} on both systems, we are able to verify Key_G each time a modifying transaction is committed locally on DB_A and DB_B .

4.3 Referential Integrity Constraints

A widely spread constraint is the definition of referential integrity on relations to specify existence dependencies between two database objects. Referring to our scenario, the company could allow researchers of department A to cooperate on shared projects of department B . Thus, we have to ensure that a researcher in DB_A is related to an existing project in DB_B and vice versa. As already mentioned, researchers are related to projects via the $prores$ relation referencing the relevant primary keys of the local project and researcher relations. Now, to reflect the global referential integrity constraint in our exemplary relational model, we allow R_B in $proresB$ to reference both, local researchers using R_B in res_B and cooperating researchers in res_A using R_A . In the scope of this paper we only consider referential integrity without cascading, although our concept of ACDBSs basically supports cascading. An outlook on cascading referential integrity can be found later in this section.

Referential Integrity Without Cascading. In the following, we focus on the referential integrity concerning the researcher number R_B in $proresB$. Referential integrity for P_B in $proresB$ is handled analogously. Similar to global key constraints, a global referential constraint is first decomposed into a set of partial constraints:

$$RefInt_G : RefInt_{resA} \wedge RefInt_{proresB}$$

The existence dependency between the local parent relation $resA$ and the local dependent relation $proresA$ can be expressed as follows:

$$\begin{aligned} local_{proresA} : \forall R, P \exists R', N, S : \\ [proresA(R, P) \wedge (R = R') \Rightarrow resA(R', N, S)] \end{aligned}$$

Furthermore we formulate a remote constraint $remote_{proresB,resA}$ as

$$\begin{aligned} remote_{proresB,resA} : \forall R, P \exists R', N, S : \\ [proresB(R, P) \wedge (R = R') \Rightarrow resA(R', N, S) \vee resB(R', N, S)] \end{aligned}$$

Using these definitions, we express the partial constraints for $RefInt_G$ as

$$\begin{aligned} RefInt_{resA} : & local_{proresA} \wedge remote_{proresB,resA} \\ RefInt_{proresB} : & remote_{proresB,resA} \end{aligned}$$

A project can only be inserted into $proresB$, if a corresponding researcher exists in either $resB$ (locally) or $resA$ (remote). Contrary, a researcher in the parent relations $resA$ and $resB$ may not be deleted, as long as depending projects exist in the dependent relation $proresB$. Thus, we have to distinguish between constraint checks for insertions and deletions on the dependent and parent relations respectively.

Insertion check: Suppose the tuple $\text{proresB}(r, p)$ is inserted, whereas p refers to an existing project in projB . According to $\text{RefInt}_{\text{proresB}}$ we have to check the existence of r locally and remote using the following tests for $\text{remote}_{\text{proresB}, \text{resA}}$:

$$\begin{aligned} \text{localtest}_{\text{RefInt}_{\text{proresB}}} &: \exists R, N, S : [\text{resB}(R, N, S) \wedge (R = r)] \\ \text{remotetest}_{\text{RefInt}_{\text{proresB}}} &: \exists R, N, S : [\text{resA}(R, N, S) \wedge (R = r)] \end{aligned}$$

If one of the tests yields *true*, then there exists a corresponding entry in either the local or remote parent relation and the tuple $\text{proresB}(r, p)$ can be inserted. Otherwise the insertion has to be rejected. For the implementation of these tests, we use the functions *checklocalkey* and *checkremotekey* as introduced in section 4.2. The remote test is evaluated using a remote state query on DB_A . A corresponding ECA rule for this partial constraint can be expressed as follows:

```
define rule PartialReferentialConstraint
on creation of a new object in proresB
if checklocalkey yields false and checkremotekey yields false
do reject transaction
```

Deletion check: Suppose the tuple $\text{resA}(r, n, s)$ shall be deleted from DB_A . According to $\text{RefInt}_{\text{resA}}$ we have to ensure that there are no depending objects in the prores relations on DB_A and DB_B before we delete this item. Thus, we formulate the following tests for $\text{local}_{\text{proresA}}$ and $\text{remote}_{\text{resA}, \text{proresB}}$:

$$\begin{aligned} \text{localtest}_{\text{RefInt}_{\text{resA}}} &: \nexists R, P : [\text{proresA}(R, P) \wedge (R = r)] \\ \text{remotetest}_{\text{RefInt}_{\text{resA}}} &: \nexists R, P : [\text{proresB}(R, P) \wedge (R = r)] \end{aligned}$$

The deletion check succeeds, i.e. $\text{resA}(r, n, s)$ can be deleted, if there are no dependent objects in proresA and proresB . This partial constraint is represented by the following ECA rule:

```
define rule PartialReferentialConstraint
on deletion of an object in resA
if checklocalkey yields true or checkremotekey yields true
do reject transaction
```

Of course, we have to ensure that an entry in the parent table exists either in resA or resB . This is realized using a key constraint on the researcher id as presented in section 4.2.

Cascading Referential Integrity Constraints. With the extended functionality of Active Component Systems, we are basically able to realize cascading referential integrity on updates or deletions of tuples. Injected transactions can be executed during the evaluation of a partial integrity constraint to modify remote data stocks including even deletions, before or after the modifying operation is committed locally. If a tuple is deleted in the parent relation, we execute an injected transaction to delete all corresponding tuples in the dependent relation. Analogously, if

a key value is updated in the parent relation, we cascade this update to the dependent relation by modifying the relevant entries in the remote database via injected transactions.

The corresponding partial integrity constraint for the parent relation is expressed similar to the partial rule without cascading presented in 4.3. We extend the rule to delete dependent objects from within the rule condition or action depending on the intended system behavior. Thus, we are able to delete entries in the dependent relation from within a *remote condition* or a *remote action*, *before* or *after* the local entry is deleted. Of course, since we modify data on more than one autonomous database system, we face the problem of atomic commitment in a multidatabase environment [10]. A distributed update may lead the federation into a (temporary) inconsistent state in case of a failure. We therefore need a recovery mechanism based on the concept proposed to detect inconsistencies and restore global integrity automatically after a transaction has violated global constraints. Following the line of argumentation in [11] we consider weakened notions of consistency, using guarantees for the level of consistency a system can provide. The integration of weakened consistency into our architecture is part of future work.

4.4 Aggregated Constraints

As a representative for this type of constraint let us assume that the company has a budget limit for all research projects. Thus, it must be checked whenever a project is created or updated in DB_A and DB_B that the sum of all project budgets B_A and B_B does not exceed a certain value ε . We restrict our further considerations on the standard aggregate functions *min*, *max*, *sum*, and *count*. The average function *avg* must be calculated during a partial constraint check using *sum* and *count*. Furthermore, we assume that $agg(T, w)$ is an aggregate function that calculates the aggregate of an attribute w of a relation T . The function $totalagg(agg(R_{m_1}, w_{m_1}), \dots, agg(R_{m_s}, w_{m_s}))$ computes the overall aggregate of partial aggregates for $m_u \in \mathcal{C}$ and $s = |\mathcal{C}|$. Please note that *count* is a semi additive aggregate function and the overall aggregate must be calculated as the sum of partial *count* aggregates.

These preliminaries provided, we can now formulate a global aggregated constraint for our example as

$$Sum_G : Sum_{projA} \wedge Sum_{projB}$$

with the partial constraints defined as

$$\begin{aligned} Sum_{projA} : totalsum(localsum_{projA}, remotesum_{projB}) &\leq \varepsilon \\ Sum_{projB} : totalsum(localsum_{projB}, remotesum_{projA}) &\leq \varepsilon \end{aligned}$$

Both databases have to check the total sum whenever an insertion or update occurs on B_A or B_B . Therefore, DB_A calculates its corresponding local and remote aggregate as

$$localsum_{projA} = sum(projA, B_A) \text{ and } remotesum_{projB} = sum(projB, B_B)$$

using two functions $agglocal : schema(projA) \rightarrow \mathbb{R}$ and $aggremote : schema(projB) \rightarrow \mathbb{R}$. The calculation of the remote aggregate is realized using a remote state query on DB_B . The aggregates for DB_B are calculated analogously.

Now suppose the tuple $projA(p, t, b)$ is inserted. According to Sum_{projA} we first compute $localsum_{projA}$ including the new value b and $remotesum_{projB}$ on DB_B . After we receive the result from the remote aggregation, we calculate $totalsum$ and compare the overall aggregate to ε . If the comparison yields *false* then the insertion of $projA(p, t, b)$ is rejected. A corresponding ECA rule for this partial constraint can be expressed as:

```
define rule PartialAggregatedConstraint
on update of  $B_A$  in  $projA$  or insertion of a new object in  $projA$ 
if  $totalsum(localsum_{projA}, remotesum_{projB}) > \varepsilon$ 
do reject transaction
```

5 Discussion

The checking mechanism presented in this paper is basically an implementation of the Local Test Transaction Protocol (LTT) presented by Grefen and Widom in [7]. The LTT exploits transaction capabilities provided by the local database system to perform a notification and wait for acknowledgment within a single transaction. Using the LTT we try to avoid remote checks by evaluating local tests first. If a local test has already failed, then we do not have to evaluate the cost intensive remote check using remote state queries. The implementation of a transaction-based protocol like LTT has to evaluate a constraint check *before* the triggering operation is committed. The external program is executed as part of the remote condition of a partial integrity rule. Our implementation certainly adopts all advantages and drawbacks of the applied LTT protocol. Thus, the implementation proposed is safe and accurate, which means that it detects all constraint violations and that, whenever an alarm is raised, there is a state in which the constraint is violated. On the other hand, since the relation is locked until the external program returns a result, the local ACDBS loses autonomy and the risk of deadlocks is relatively high, if relations in DB_R and DB_S are updated concurrently.

Due to the flexibility of our architecture, we are basically able to implement the entire set of protocols described by Grefen et al. Thus, to overcome the drawbacks of the LTT, we can modify the partial integrity constraints to implement the Materialized Delta Set Protocol (MDS), which increases autonomy and reduces the risk of deadlocks. Therefore, we maintain an additional relation ΔR , which stores an accumulated set of updates of the original relation R . The constraint checking mechanism is then evaluated using ΔR instead, so the original relation is not locked during the check. This enables at least concurrent read access to R while updates must still be delayed until the lock is released. In our architecture ΔR is maintained using the active capabilities of the DBMS. We define a local rule on R to copy all updated items to ΔR . The partial integrity rule including the remote checks as presented above is then expressed over the Materialized Delta Set ΔR .

A generalization of the constraint checking mechanism to more than two sites is tightly corresponding to the implemented integrity checking protocols. As already described in [7], most of the constraints that involve more than one site can be decomposed into a couple of constraints, which are expressed over exactly two databases. If there is the need for multi-site constraints, the authors propose to use non-transaction-based protocols like Direct Remote Query (DRQ) or Timestamped Remote Query Protocol (TRQ), which can both be implemented using our architecture. To avoid locking of the updated relation we adjust the partial integrity rule to be evaluated *after* the modifying operation is committed.

The protocols could be optimized according to the Demarcation Protocol presented in [12]. This protocol is particularly suitable for arithmetic constraints like aggregated constraints, but can also be used for key or referential integrity constraints. The Demarcation Protocol can be seen as an extension to the LTT and thus be implemented using our architecture.

6 Related Work

In the last years, research on integrity constraints in heterogeneous environments mainly considered the simplification, evolution, or reformulation of constraints rather than mechanisms or protocols for integrity checking. A closely related concept in terms of the rule structure and constraint types is presented in [9]. The authors use private and public global constraints to define dependencies between data in different databases, similar to the partial constraints presented in this paper. One of the main differences is the use of a layered approach to support the active functionality required for event detection and rule processing. A reactive middleware based on CORBA encapsulates active and passive sources and processes rules using an external remote rule processing mechanism. Furthermore every component is assumed to have an Update Processor to execute local update requests, but the local relation cannot be locked during the evaluation of remote conditions.

The metadatabase approach [13] uses a rule-oriented programming environment to implement knowledge of information interactions among several subsystems. Each subsystem is encapsulated by a software shell, which is responsible for monitoring significant events, executing corresponding rules, and interacting with other shells. Although conditions can be evaluated in a distributed way, the rule processing itself is still centralized.

A distributed rule mechanism for multidatabases is presented in [14] as part of the Hyperion project. A distributed ECA rule language is introduced, which is mainly used to replicate relevant data among data peers in a push-based fashion. Rules are processed by a rule management system that resides in the P2P layer on top of a peer database. The X^2TS prototype [15] integrates a notification and transaction service into CORBA using a flexible event-action model. The architecture presented resembles a publish/subscribe system, whereas publishing of events is non-blocking. Another middleware approach for distributed events in a heterogeneous environment is presented in [16]. CORBA-based, distributed, and heterogeneous systems are enhanced by Active DBMS-style active functionality. The ar-

chitecture uses wrappers with event monitors to detect data modifications in the data source.

A common characteristic of the architectures just mentioned is the use of a layered approach with event monitoring to somehow notify a mediating component (e.g. a constraint manager, rule processor, or middleware component) about events occurring in the local database. If the source is not monitored, the notification mechanism is generally based on active capabilities of the underlaying database management system, but there is so far no detailed description of this interaction published.

The most distinctive characteristic of our concept is the direct usage of existing active capabilities of modern database management systems without the need for wrappers or monitoring components. Since a remote condition is evaluated during the execution of a trigger, it is irrelevant if the triggering transaction was a global or local update. We benefit from the active functionality of the DBMS in terms of transaction scheduling, locking, and atomicity, resulting in a synchronous integrity checking mechanism. Especially the ability to rollback updates depending on a remote state query makes corrective or compensative actions basically superfluous.

7 Conclusion and Future Work

We have presented an architecture for global integrity maintenance in a federated relational database using component database systems with enhanced active functionality. We introduced Active Component Database Systems which are able to communicate with other component databases to which their data is semantically related to. They are no longer just passive data providers but actively participating in global integrity maintenance. Global integrity constraints are composed of sets of partial integrity constraints for each component system that is affected by the constraint. The partial constraints are evaluated using local and remote checks which are implemented entirely on a local site. We have described the requirements and basic functionality of our architecture and provided examples for partial constraints for commonly used classes of global constraints.

In the next steps we address the problem of distributed updates using injected transactions as needed for cascading referential integrity or data replication. Furthermore, we plan to deal with the detection, resolution, and prevention of deadlocks and examine the system behavior depending on the time a rule is evaluated and an external program is called. The architecture shall be elaborated in a true loosely coupled environment to evaluate execution costs and scalability.

References

1. Heimbigner, D., McLeod, D.: A Federated Architecture for Information Management. *ACM Transactions on Information Systems (TOIS)* **3** (1985) 253–278
2. Sheth, A.P., Larson, J.A.: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys* **22** (1990) 183–236

3. Paton, N.W., Díaz, O.: Active Database Systems. *ACM Computing Surveys (CSUR)* **31** (1999) 63–103
4. Loney, K., Koch, G.: Oracle8i: The Complete Reference. Osborne/McGraw-Hill (2000)
5. Popfinger, C., Conrad, S.: Tightly-coupled Wrappers with Event Detection Subsystem for Heterogeneous Information Systems. In: DEXA Workshop Proceedings, IEEE Computer Society Press (2005)
6. Chamberlin, D.: A Complete Guide to DB2 Universal Database. Morgan Kaufmann (1998)
7. Grefen, P.W.P.J., Widom, J.: Integrity Constraint Checking in Federated Databases. In: Conference on Cooperative Information Systems, IEEE Computer Society Press (1996) 38–47
8. Türker, C., Conrad, S.: Towards Maintaining Integrity of Federated Databases. In: Data Management Systems, Proc. of the 3rd Int. Workshop on Information Technology, IEEE Computer Society Press (1997) 93–100
9. Gomez, L.G.: An Active Approach to Constraint Maintenance In A Multidatabase Environment. PhD thesis, Arizona State University (2002)
10. Mullen, J.G., Elmagarmid, A.K., Kim, W., Sharif-Askary, J.: On the Impossibility of Atomic Commitment in Multidatabase Systems. In: Proc. of the 2nd Int. Conf. on System Integration, IEEE Computer Society Press (1992) 625–634
11. Chawathe, S., Garcia-Molina, H., Widom, J.: A Toolkit For Constraint Management In Heterogeneous Information Systems. In: Proc. of the Int. Conf. on Data Engineering. (1996) 56–65
12. Barbará-Millá, D., Garcia-Molina, H.: The Demarcation Protocol: A Technique for Maintaining Constraints in Distributed Database Systems. *The VLDB Journal* **3** (1994) 325–353
13. Hsu, C., Rattner, L.: Metadatabase Solutions for Enterprise Information Integration Problems. *DATA BASE* **24** (1993) 23–35
14. Kantere, V., Mylopoulos, J., Kiringa, I.: A Distributed Rule Mechanism for Multi-database Systems. In: CoopIS/DOA/ODBASE. (2003) 56–73
15. Liebig, C., Malva, M., Buchmann, A.P.: Integrating Notifications and Transactions: Concepts and X^2TS Prototype. In: EDO. (2000) 194–214
16. Koschel, A., Kramer, R.: Configurable Event Triggered Services for CORBA-based Systems. In: EDOC. (1998) 306–318

RFID Data Management and RFID Information Value Chain Support with RFID Middleware Platform Implementation

Taesu Cheong and Youngil Kim

Electronics and Telecommunications Research Institute, 161 Gajeong-dong,
Yuseong-gu, Daejeon, 305-700, Republic of Korea
{qlink, embroca}@etri.re.kr
<http://www.etri.re.kr>

Abstract. Radio Frequency Identification (RFID) middleware is a new breed of software system which facilitates data communication between automatic identification equipments like RFID readers and enterprise applications. It provides a distributed environment to process the data coming from tags, filter and then deliver it to a variety of backend applications via various communication protocols including web services. In this paper, we focus on the information flow converting raw RFID data to useful information which may even be used to lead to automated business process execution and further to knowledge to support decision making, and we define the information flow as so-called ‘RFID Information Value Chain (RFID IVC)’. We examine the elements and associated activities of RFID IVC and also introduce the RFID middleware ecosystem not only to provide the seamless environment spanning from the edge of the enterprise network to the enterprise systems, but also to support the activities arisen on RFID IVC. RFID middleware ecosystem consists of RFID middleware, rule engine to generate business semantic events and orchestration engine to coordinate sort of business process invoked by RFID tag data capture event. Moreover, the implementations of each system residing in RFID middleware ecosystem are introduced and the relationship between RFID middleware ecosystem and RFID IVC is demonstrated.

1 Introduction

Along with the rapid evolution of Internet and system integration technologies, research direction of ubiquitous computing introduced by Mark Weiser in 1988 has moved from an imaginary phase into a realizable research phase. To accomplish the ubiquitous computing environment, many research activities have been going on within many forms of technology domains including home networking, wireless sensor networking, telematics and so on. Especially, RFID technology that uses radio waves to automatically identify people or objects by attaching RFID tags storing serial numbers to them has recently gained a lot of attention and the RFID market have made substantial progress from standards development to expanded pilots to real deployments. Currently, the most well-known technology to identify physical objects is barcode, but it needs human intervention. Compared with the barcode system, RFID system has many benefits – such as no human intervention, the identification of

multiple objects simultaneously and so on – which in turn may make a breakthrough in business for automation. For example, business benefits will include better inventory management across the supply chain, labor efficiency and enhanced product integrity.

So far, the areas related to RFID hardware devices like tags and readers have mainly dominated the RFID market and research themes. However, the trend has gradually shifted to the RFID software system as the necessity for processing huge amount of the data streams delivered by RFID hardware devices gets larger. The focus on hardware is important, but RFID hardware is of minimal value without effective software that can aggregate data from RFID readers and pass it to enterprise applications. The basic functions of RFID software start with device monitoring and management. It extracts data from readers, filters and aggregates the information, and then sends it to enterprise system. Moreover, the market requires a kind of middleware software platform that provides the environment that information about the identified tag is shared with internal or external applications.

In this paper, we propose the concept of ‘RFID Information Value Chain (RFID IVC)’ which describes the data transformation process converting raw RFID data to useful information and further knowledge that are significant to decision making in the business context under the RFID-based environment. For this, the associated activities for each stage of RFID IVC and the specific data processing methods are discussed. Moreover, we present RFID middleware platform to support the RFID IVC and introduce the prototype systems which are based on the proposed platform.

The remainder of this paper is organized as follows. In the section 2, we briefly provide a literature survey on RFID system itself and a reference model for an RFID software system. Section 3 introduces the concept of ‘RFID Information Value Chain (RFID IVC)’ and discusses the details of the data transformation process from data to information and knowledge. In section 4, we present the RFID middleware platform in order to support the RFID Information Value Chain and introduce the system implementation of RFID middleware platform called ‘ETRI RFID Ecosystem’. The final section summarizes and presents the future works.

2 Radio Frequency Identification (RFID) System and EPC (Electronic Product Code) Network

RFID is the combination of radio technology and radar that detects the given object through the reflection of radio waves. Generally, a basic RFID system consists of three components as follows: a RFID tag which is attached to the object to be identified and serves as data carrier, a RFID reader which is used to read and/or write data to RFID tags, and software systems including RFID middleware. Figure 1 shows RFID system components according to the report by Accenture [1].

Here it should be noticed that, we assume, any detailed or historical information of the given object to which a RFID tag is attached should be stored and managed in an external data storage connected to the network, because the memory size of a RFID tag, particularly for the passive tag, is often limited (mostly due to its cost or the whole tag size). And RFID middleware software should provide functionalities for an

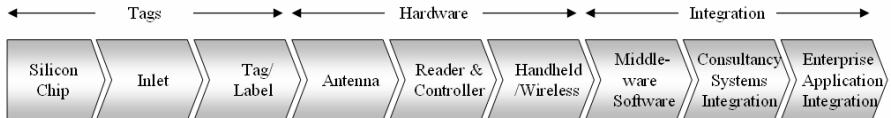


Fig. 1. RFID Solution Components. The RFID solutions consist of three major parts: a tag associated with the object to be identified, hardware including a reader and antennas, and software integration.

efficient distribution of information through the network infrastructure. Thus, for that reason, middleware software in the RFID system is considered as a kind of ‘broker’ for integrating RFID reader devices with the back-end applications.

In regards to a middleware in the RFID system, however, there are few ongoing activities except those by EPCglobal, Inc. (also named shortly EPCglobal) [2], which is a standards management and development body chartered with defining the de-facto standards related to RFID system. EPCglobal mainly works on RFID applications in domain of supply chain and logistics industry from all angles including businesses, technologies, and policies. Moreover, EPCglobal proposed two interesting concepts; one is electronic product code (EPC) which is a numbering scheme for uniquely identifying object, and the other is EPC Network, as shown in Figure 2, which comprises several functional roles and interfaces based on RFID system and information.

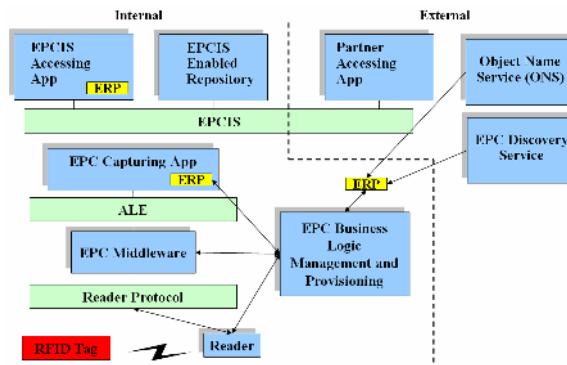


Fig. 2. The figure shows how EPC Network presented by EPCglobal is organized [3]

Particularly, the researches about middleware in EPCglobal have been continued with somewhat gradual progresses. In the initial specifications from Auto-ID Center [4], the middleware, called ‘Savant’, was to perform data routing operations for data capturing, monitoring, and transmission, and it consisted of three major internal components in a hierarchical tree structure; event management service (EMS) that collects and processes tag-read events, real-time in-memory data structure (RIED) that is a temporary data storage of tag-reads and task management service (TMS) that performs customizable tasks to maintain “Savant” itself.

In the later version of the specification about savant from EPCglobal [5], a matter of concern moved from the specific processing features into a flexible container with the generalized external interfaces for outer service applications. Savant is a container of processing modules for specific features and may be customized to meet the needs for applications. Each processing module including EMS, RIED and TMS may interact with outer services via two predefined interfaces – that is, Reader Interface for communicating with RFID readers and Application Interface for providing interaction with external applications. Most of commercial RFID middleware software systems which are currently available in the market - such as Oracle [6], Sun Microsystems [7] and so on - follow the ‘Savant’ structure mentioned so far.

In the latest version about the middleware, now called ‘Application Level Events (ALE)’ [8], its purpose is to process the reduction of the data volume directly coming from various sources such as RFID readers and to route the events of interest to applications. ALE provides interfaces for defining the control and delivery of the filtered and collected tag read data, and then outer applications have access to ALE in order to obtain the tag read data of interest.

So far, it is presented how the middleware concept developed by EPCglobal has been evolved. As mentioned before, most of currently market-available middleware software systems follow EPC network architecture spanning not only middleware itself but also so-called EPC Information Service (EPCIS), sort of distributed networked database managing EPC-related data including product-specific data and their trace over supply chain. However, EPC network architecture is mainly focused on SCM (Supply Chain Management) – oriented mechanism and associated information traceability. In this paper, we view the RFID-based system architecture from a different standpoint – that is, the data transformation process converting from raw data into valuable information – and propose RFID system framework supporting such the data transformation process as well as RFID-based automated business process execution in the end.

3 RFID Information Value Chain and Multi-stage RFID Data Processing

In this section, we discuss how to manage the information flow from the recognition of RFID tag to the delivery to the applications that consume RFID information over the RFID solution components introduced in Section 2. As already mentioned in the previous section, the RFID system generally focuses on the automatic identification of individual objects without human intervention. Moreover, the structure of EPC Network proposed by EPCglobal is designed to support the information exchange among trading partners over the supply chain. However, in order to create big value-add by utilizing RFID infrastructure, the following process must go through: The information that is filtered and summarized after RFID readers capture raw RFID data is well-used for the enterprise applications as valuable resources, and later significant knowledge is derived from the accumulated information. Low-value raw RFID data is transformed into useful information and further guidance for actions. This means that the flow of information is a value-added process. In this paper, we define this value-added data transformation process as ‘RFID Information Value Chain (RFID IVC)’

and it pursues the maximization of the benefits from the RFID deployment. Figure 3 shows RFID IVC along with the RFID solution components shown in Figure 1 and the correspondent activities for each step are described. In general, the information value chain can be defined as the flow from raw data to information and further to knowledge, and the same rule can be applied to the RFID IVC.

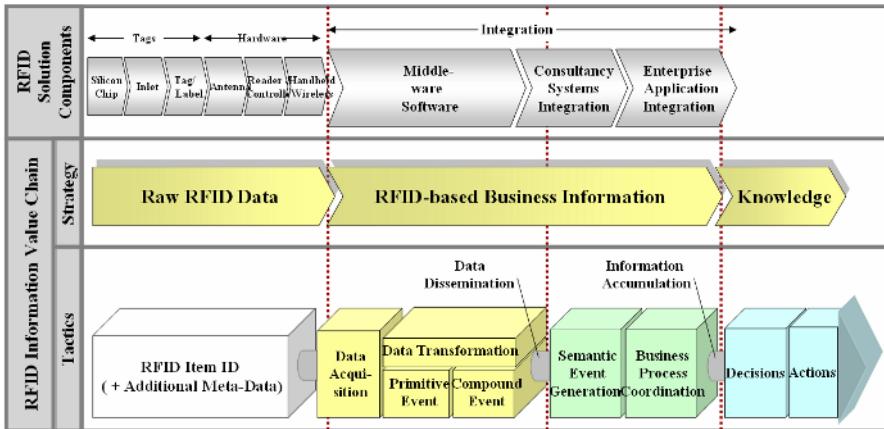


Fig. 3. RFID Information Value Chain (RFID IVC). This figure shows how the raw RFID data is transformed into information and further to valuable knowledge.

In this paper, we specify the associated activities occurred in each stage of transformation process and the details are shown in the lower most part of Figure 3. Here after, the derived stages and activities for each stage are demonstrated as follows.

3.1 Data Acquisition

This stage is to collect the RFID reads from various kinds of RFID readers and other types of object identification devices like barcode system. At the present, there exist different kinds of RFID tag-reader protocols including EPCglobal Class 0, Class 1, and Class 1 Generation 2 tag-reader protocols [9], and ISO/IEC 18000-x series [10]. Moreover, the market-available reader systems use the different I/O protocols such as RS232C, TCP/IP and USB for the communication between the RFID reader and the host systems. At this stage, the situation that the heterogeneity among RFID reader systems exists must be dealt with in order to support the seamless data in-flow.

3.2 Data Transformation

Usually, RFID data in the form it is acquired has little direct value to the enterprise applications. To have value, it must be transformed into a more useful form. This stage, data transformation, primarily involves either filtering or aggregation of raw RFID data. Here, the meaning of ‘filtering’ is that not all data may be necessary to the applications, so significant data items are identified and the remainder discarded.

With the consideration of the characteristics of RFID, the tag data sensed by RFID readers have the technical limits as follows:

First, RFID readers cannot guarantee 100% accuracy of tag reading at the present because of interference and high sensitivity by the surrounding environment. Therefore, it is necessary to resolve the physical limits and the smoothing process can be considered as one method [11].

Second, RFID reader is physically non-contact to communicate with tags and the number of RFID tag data flowed from a RFID reader ranges from 10s of tag data per second up to more than 100s a second. This leads to bring the big burden to the host system which is responsible to process all the data, so the appropriate scheme for data volume reduction is required.

Third, as pointed out at the second, an RFID reader can read multiple RFID tags simultaneously and, sometimes, tags are unintentionally sensed from an area beyond the area intended to be monitored by the reader due to many reasons including the reflection of radio wave. Among the sensed tag reads, not all the data are required to the applications that consume RFID data, so the tag data in which the applications are not interested should be filtered out.

This stage is responsible to handle the problems mentioned above and, here, two levels of RFID event processing are considered: primitive and compound event processing.

Primitive Event Processing

The data emerging directly from RFID readers may be regarded as a stream of records of the form (r, n, g, t) , denoting the antenna n of the reader r reads tag g at time t . In this stage, it is responsible for mapping the low-level data stream having the limited information to more manageable form that is suitable for application-level interactions.

As the first step to support the data transformation, one of the main objectives on this sub-stage is to reduce the volume of data stream by discarding the redundant tag data. In general, when a tag appears present to a particular RFID reader for many read cycles, this generates a lot of data. Moreover, the tag might not appear every read cycle although the tag stays in the read range. As the methods to help overcome these problems, the event smoothing process can be introduced.

The meaning of ‘smoothing’ here is to pass the tag read only when something of interest happens such as when a tag is first sensed by the reader or when the tag is no longer present. The state diagram in Figure 4 shows how the smoothing process works.

As seen in Figure 4, three states per each tag – Unknown, Sensed, Captured – and four different events – eventSensed, eventDisappeared, eventCaptured, eventExpired – are introduced and the tag read is passed to next stage only if the state transition between two adjacent states occurs. Initial state of a tag is ‘Unknown’ and, when a tag appears for the first time in the read range, the event ‘eventSensed’ is generated and the current tag state is moved onto the state ‘Sensed’. The event ‘eventCaptured’ is generated when the tag is seen for a certain period and the event ‘eventExpired’ occurs if current state of the tag is ‘Captured’ and has not seen for a while. The event ‘eventDisappeared’ is generated when the tag hasn’t seen for a time without subse-

quently generating ‘eventCaptured’ event. If a tag generates ‘eventSensed’ and then ‘eventDisappeared’, it means that the tag enters the read range briefly and it can be regarded as a candidate for an inadvertent tag read.

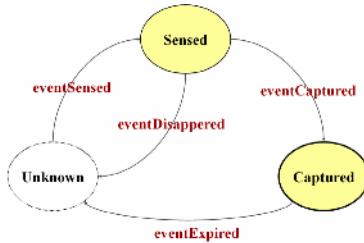


Fig. 4. Primitive RFID Event Generation. We simplify the event state transition scheme proposed by Auto-ID Center [11].

What we can gain from this primitive event processing sub-stage is as follows: (a) reduction of data volume by reporting only if the state transition occurs, (b) establishment of the decision basis which is used to distinguish tags that remains for time interval enough to be regarded as tags that the reader should monitor, as compared to tags that enter the read range only briefly or are reported unwillingly.

Compound Event Processing

As mentioned before, the primitive event processing sub-stage is mainly focused on the duplicated data volume reduction. On the other hand, this stage, compound event processing, concentrates on the selection of tag data that are meaningful to the business domain while having the limited information delivered by RFID reader.

Data from readers follow the feature of record (r, n, g, t) and, as passing through the primitive event generation process, the event type e - which itself can be regarded as significant event information - is added as an additional field in the record as like the record (r, n, g, t, e) in return for the elimination of redundant data. At this stage, the useful information has to be derived from the restricted information.

Basically, the event filtering can be applied to each field of the record or the combination of fields. For example, an application may want to select the tag reads which are captured by specific readers and another application may be interested in the tag reads that their IDs start with the specific pattern. Here, we have five categories to apply the filtering operation – reader ID, antenna number, tag ID, timestamp and event type – and those are the basic filtering schemes. The basic filtering schemes based on the following fields – ‘reader ID’, ‘antenna number’, ‘tag ID’ and ‘event type’ – are applied like this: the filtering operations applied to the associated field on the event record (r, n, g, t, e) forward the records that only fit within specific range or pattern; on the other hand, the basic filtering scheme based on the field ‘timestamp’ can be utilized as duration-based filtering. The timestamp-based filtering scheme usually combines with other basic filtering schemes to support more intelligent filtering.

In addition to the basic filtering schemes, many other filtering schemes are considered as follows:

Association. In the case that several readers are associated with the same tag reads, it needs to allow a group of readers to be filtered for duplicate RFID tags so that the application which consumes the tag read seems as if the tag read is originated from the single reader. To provide the association, it needs to combine the event record (r, n, g, t, e) with the relationship (r, l) providing the logical location l of reader r .

Batch. In some cases, a number of tags is sensed by readers at the same time, then the group of sensed tags itself seems significant event to the applications. For example, suppose that readers located in the entrance of warehouse capture a number of cases and one pallet containing them within a short time interval. In order to derive the meaningful information to the applications, the sensed tag list including both cases and a pallet should be processed as a batch. That is, this batch filtering scheme clusters tag reads into groups of distinct tags based on when they have been observed during the given interval.

Read-Range In/Out. For the situation of warehousing of goods or taking them out of the warehouse, reporting in-field and out-of-field events when tags move in and out of the read range is more significant to the associated applications that are responsible to process business based on the tag reads. This filtering scheme can be solved by making use of the ‘event type’-based basic filtering scheme; that is, use the event type ‘eventCaptured’ and ‘event Expired’ respectively among the fields of the record (r, n, g, t, e).

In general, the filtering schemes which applications want to apply vary from application to application and, we believe that the schemes discussed above are considered as the commonly used filtering schemes.

3.3 Data Dissemination

A critical task of any information system is to deliver the right information to the right people or applications at the right time. The dissemination activity involves moving the filtered and aggregated information from RFID readers to enterprise applications or other business integration applications. The purpose of data dissemination activities is to determine who needs what information and to deliver it to them on time.

In order to meet this requirement, we provide two types of data dissemination models; the push model that RFID data captured by RFID readers keep flowing upward to the back-end applications over pre-defined event pipeline, and the pull model (in other words, publish/subscribe model) that applications which are interested in handling RFID data subscribe to the middleware system with additional information such as notification cycle, reader set which they are interested in listening to and so on, and the middleware plays a role of dispatching messages to the subscribers asynchronously.

3.4 Semantic Event Generation

At the stage 2 of ‘data transformation’, filtering and aggregation methods of raw RFID data are demonstrated. However, it may not be sufficient for the filtered data to become the beneficial information to enterprise applications. At this stage, the filtered

data are combined with different sources residing in the legacy system or external sources such as purchased data services so that semantic events which are significant in the business domain are generated. Here, the term “semantic event” means that, as it says, RFID data capturing events merely indicate raw observation and taking business actions based only on the event reports are somewhat limited; therefore, the raw observation events need to be combined with additional business context information in order to construct business events (here, we call ‘semantic event’) upon which legacy applications can act. Furthermore, the reason why this stage is required is to enable sophisticated RFID-based data processing. As domain-specific information is integrated with RFID tag data, content-based filtering and routing become possible.

As the basic approach, we can get the tagged object information whose ID is matched with the tag ID passed from the previous stage. From them, so-called content-based filtering can be applied. For example, we can filter out the product information that its price is lower than \$5.

In other way, Event-Condition-Action (or ECA) rules can be used to generate the semantic events to trigger the relevant business processes. When a primitive tag list is delivered from the previous stage as an event, the rule set associated with the event is evaluated and then appropriated actions are taken and executed. We can take advantage of the ECA rule mechanism when predefined action is taken by the comparison of the sensed tag list with the scheduled information. If the sensed tag list mismatches the schedule information, another action to detect the problem will be taken. We can say that the inference process is regarded as the semantic event generation making use of RFID tag data and additional information stored in backend systems and supports the conversion from raw RFID data to actionable information.

3.5 Business Process Coordination

At this stage, its main objective is to enable business processes and solutions to leverage the real-time data captured by RFID infrastructure. The key benefit of RFID technologies is automatic identification of individual objects coupled with automatic data capture. From the employment of low-levels of process automation to the process automation and efficiency improvement ultimately lead to the high return in terms of efficiency and cost reduction.

3.6 Decision / Actions

One of the expected benefits from RFID deployment is real-time information gathering and real-time item visibility. It means that real-time decision making would be realizable. For example, real-time inventory monitoring suggests optimum reorder points based on usage and improves inventory accuracy. Another purpose of this stage is to provide guidance for action to decision maker based on the accumulated information and ultimately produce the knowledge. As a lot of RFID data and related production information are accumulated, it is possible to elicit the valuable knowledge from them. There are many methods to produce guidance for decision maker and further knowledge as following:

Views of current or historical information. This is the simple approach and the modeling usually consists of aggregation, summarization and filtering.

Forecast. This requires using a methodology like statistical regression based on the current and historical information.

Recommendation of the best and alternative decisions. To find the best recommendation, an optimization model searches among various alternatives and decides the best. Finding a reorder point in inventory problem through various optimization techniques is an example.

Inference through data mining. This is the process to elicit knowledge by searching for the pattern hidden within accumulated information.

4 RFID Middleware Platform for the Support of RFID Information Value Chain and Its Implementation

So far, the concept of RFID IVC is introduced and the detail activities for each stage of RFID IVC are demonstrated. In this section, we introduce the RFID middleware platform which meets the general RFID middleware requirements and supports the RFID IVC discussed in Section 3. Moreover, the platform implementation we have worked on will be presented.

4.1 RFID Middleware Platform: Introduction

RFID middleware is a software system that facilitates data communication between automatic identification equipment such as RFID readers and enterprise applications. There are many capabilities in order to build up RFID software system and, especially, RFID middleware platform capabilities include:

Reader Management. RFID middleware should support means to deploy, monitor and issue commands to readers via a general interface.

Data Management. As tag reads are flowed into the middleware, there are a lot of noises and duplicate reads, so filtering operation is required to eliminate such information that is either redundant or unnecessary.

Application Integration. RFID middleware delivers the tag reads to back-end system - for example, SCM, ERP or WMS, etc - for the better business decision.

Rule and exception processing. RFID middleware platform provides the environment that users set the rules in order to generate the business dependant semantic events based on the tag reads.

Process Management. To be an intelligent and sophisticated RFID middleware platform, it not just routes RFID data to enterprise applications but also actually orchestrates RFID-related end-to-end processes that touch multiple applications or legacy systems.

Share of data with partners and other applications. One of the ultimate goals by adopting RFID technology is to provide the ways to share the information about individual tagged object and related business descriptions with the trading partners and other applications

Among the six capabilities, first three capabilities – reader management, data management and application integration – are the MUST-DO features to be an RFID middleware [4][5]. On the other hand, the last three capabilities are essential to build up concrete RFID-based software platform.

We propose ‘ETRI RFID Ecosystem’ to support not only the presented capabilities that RFID middleware platform must provide but also the activities occurred on RFID IVC, and ultimately provide the seamless environment spanning from the edge of the enterprise network to the enterprise systems. ETRI RFID Ecosystem is a multi-layered middleware platform in Java environment. The first layer – RFID Event Management System (REMS) – deals with raw-level data filtering and aggregation. The second layer – Real-time Business Process Triggering System (RBPTS) – has the responsibility to generate the semantic business event by utilizing filtered RFID data. The next layer above – Orchestration Engine (OE) – is to support the autonomous business process execution. The top layer deals with the generation of invaluable knowledge. Additionally, Tagged Object Information Repository manages the tagged object information and makes them available to whatever the information are required. The following section discusses the system architecture and implementation issues per each system shown in Figure 5.

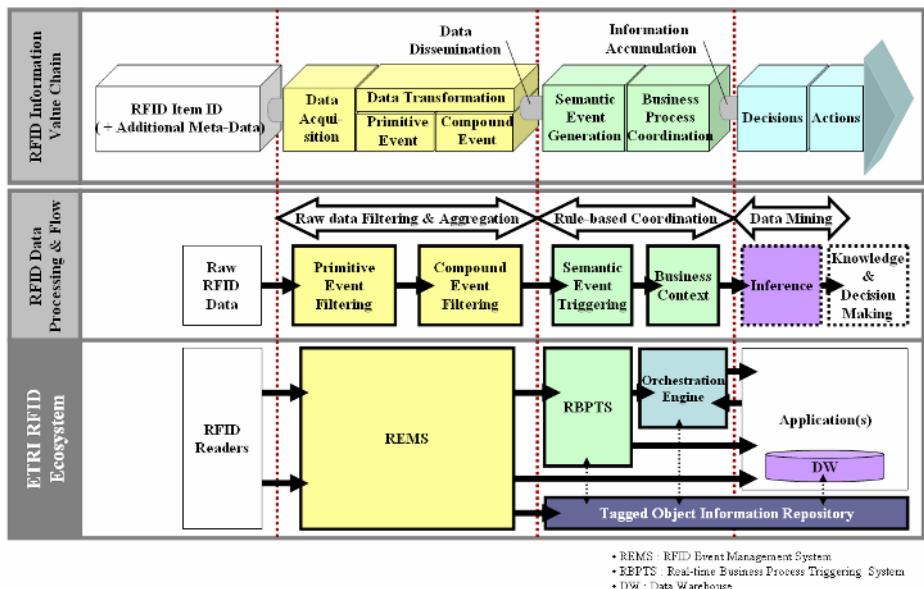


Fig. 5. ETRI RFID Ecosystem and RFID IVC. This figure introduces RFID middleware platform we propose – so-called ‘ETRI RFID Ecosystem’ – and presents the relationship between RFID IVC and ETRI RFID Ecosystem.

4.2 ETRI RFID Ecosystem: RFID Middleware Platform Implementation

ETRI RFID Ecosystem is an RFID middleware platform we propose and it contains three primary components; that is, RFID Event Management System (REMS), Real-

time Business Process Triggering System (RBPTS) and Orchestration Engine (OE). It is designed to offer the seamless environment extending from RFID hardware infrastructure to back-end software system and support the RFID IVC. In this section, the functional features and the architecture of each system that constitutes ETRI RFID Ecosystem will be described in the following.

RFID Event Management System: RFID Middleware

Basically, the proposed RFID Event Management System (REMS) is layered into several levels spanning from the device layer up to the legacy application layer as shown in Figure 6 and, at each layer, a number of components need to be defined. These components are discussed in the following:

RFID Device Abstraction Layer. This layer abstracts the tag-read protocols, supports various types of reader devices and allows users to configure, monitor and control all these devices. This layer consists of four modules – reader profiler, protocol processor, command processor and reader monitor. Reader Profiler manages and maintains the data about the devices which are deployed to the middleware system. In the middleware, two commercial reader systems including Alien[12] and Symbol[13], a passive RFID reader that ETRI developed and is compatible with EPCglobal, and an active reader that ETRI developed and is compatible with ISO 15961 standard are supported. Protocol processor helps ensure the middleware has access to the RFID readers via various I/O communication options including TCP/IP and RS232C, and support low-level integration with the hardware devices. Its purpose enables readers from many different manufactures to interact with the middleware application with the seamless way. Command Processor converts the commands issued by users to reader-aware commands and then passes them to Protocol Processor. Reader Monitor is used to monitor the healthy of the deployed RFID readers and manage the readers through Graphical User Interface (refer to Figure 7).

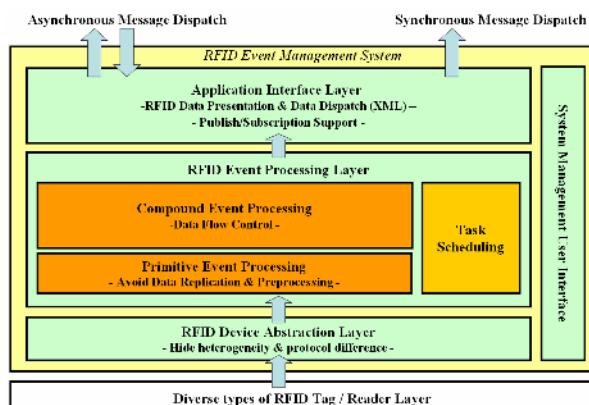


Fig. 6. Architecture of RFID Event Management System (REMS). REMS is a core of RFID middleware platform. It sits between RFID readers and the applications in RFID system and provides the following stages of RFID IVC – data acquisition, data transformation and data dissemination.

RFID Event Processing Layer. This layer is correspondent with data transformation stage of RFID IVC. It performs filtering, aggregation, and routing the RFID event data coming from the lower layer. The number of RFID event data coming from the device abstraction layer ranges from 10s of event per second up to more than 100s a second, so it is important to apply appropriate filtering processing discussed in section 3.2 on them. Filtering of that data can eliminate such information that is either redundant or that the client applications are not interested in. The filtering requirements mainly depend on the application domain. Some of filtering examples that might be required of almost all applications are following. Basically, a reader reports redundant tag reads to applications as long as a tag stays in the region. A primitive event processing is to eliminate the redundant tag read events. In some cases, a tag cannot be seen for every read cycle of a reader because the RFID reader cannot report the tag reads with 100% accuracy. Besides, unwilling tag read is sometimes reported in case that an unexpected tagged object bypasses near the region. Coping with all the situations, so-called event smoothing process is applied and it plays a role of event pre-processing at the primitive event processing stage. After passing through the primitive event processing stage, it is moved onto the compound event processing stage. The purpose of this stage is to discard uninterested tag reads. One example is the ID-based filtering. Every tag has its own unique ID and, if ID matches the predefined bit pattern, then the filter passes it to applications while remains are thrown away. Also, filtering based on reader identity is one of significant filtering techniques. Lastly, multiple readers can report the same tag read if they are placed close to each other and they are related to the same semantic operation like readers to capture the entering of products in a warehouse. For that case, association filter is used to select a tag reads among multiple tag reads from readers. All the filters discussed so far are supported by our middleware software as built-on filters. Moreover, to meet the application requirements, REMS provides user interface to build up a chain of filters as shown in the right of Figure 7. We call the chain of filters ‘Event Pipeline’. Event Pipeline Designer in Figure 7 displays the registered filters and allows users to model the pipeline which describes how the event message can be filtered, buffered and delivered to applications. All the event filters are connected in directed acyclic graph fashion to

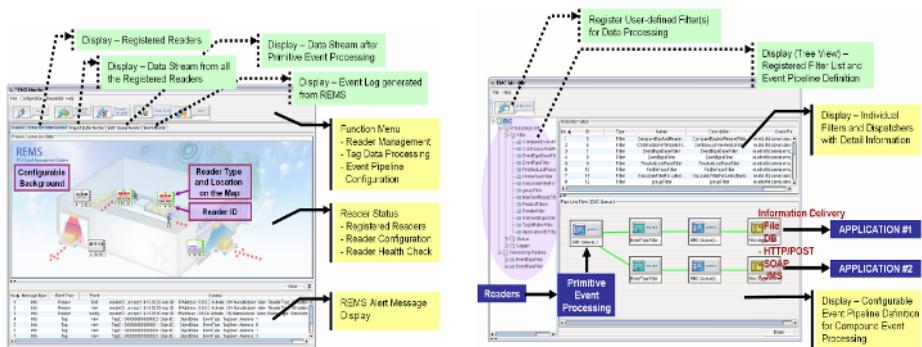


Fig. 7. Implementation of REMS: (a) GUI window of Reader Management (b) GUI window for managing customizable event filters and building ‘Event Pipeline’

meet the needs for the domain. The designer component saves the flow diagram which is the result of the modelling as a XML format file. Compound Event Processing module instantiates and executes the event pipeline by interpreting the XML configuration file.

Application Interface Layer. This is the layer for applications to get the filtered and aggregated RFID data and we provide (a) synchronous message dispatching (push model) and (b) asynchronous message dispatching (pull model). The former means that it sends the filtered RFID data to the applications whenever the data is delivered after filtering. We prepare a set of ready-to-use message dispatchers including: File message dispatcher that writes tag list into specified file, Database message dispatcher that inserts a tag data into designated database, HTTP message dispatcher that sends XML-encoded tag list via HTTP/POST, and SOAP message dispatcher that sends XML document typed tag list over SOAP protocol. Here, XML document follows PML-core Specification [14]. We also provide user interface to register custom message dispatchers which follow the application-specific protocols. On the other hand, asynchronous message dispatching is realized by providing the SOAP APIs following publish/subscribe manner. First, the application which is interested in receiving tag list registers URL with access information representing the receiver. When registering URL, the application specifies the notification cycle. The middleware then accumulates the tag received according to the notification cycle and notifies the tag list to URL that the application specified.

Real-time Business Process Triggering System: Rule-based RFID Event Processor

Real-time Business Process Triggering System (RBPTS) is built on a rule engine and its use is to generate the domain specific semantic event using a set of rules defined by domain experts. The semantic events – as discussed in section 3.4 – are produced by associating the RFID primitive events with the domain-specific information residing in legacy system. This system receives a continuous stream of filtered and unfiltered RFID data from RFID middleware or RFID readers and produces the

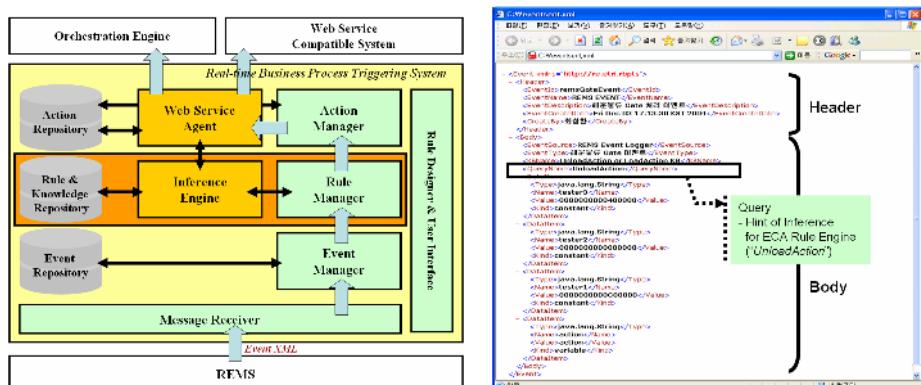


Fig. 8. Architecture of Real-time Business Process Triggering System and Sample Event XML Message over SOAP/HTTP delivered from REMS

RFID-related significant business event by using set of rules modeled by business experts. The produced semantic event is used as the query to execute collection of rules to perform various predefined actions ranging from one-time actions such as DB operation, the notification, alerts, actuator operations, or actions that involve the long term business process actions which require interoperation with workflow systems such as ebXML engine and BPEL-based workflow engine.

The development of RBPTS is driven by the requirement of flexible way of incorporating RFID data with business applications; that is, to convert the data from lower RFID middleware layers to actionable semantic information for the upper layers; that is based on the business or process semantics as perceived by the user of the information.

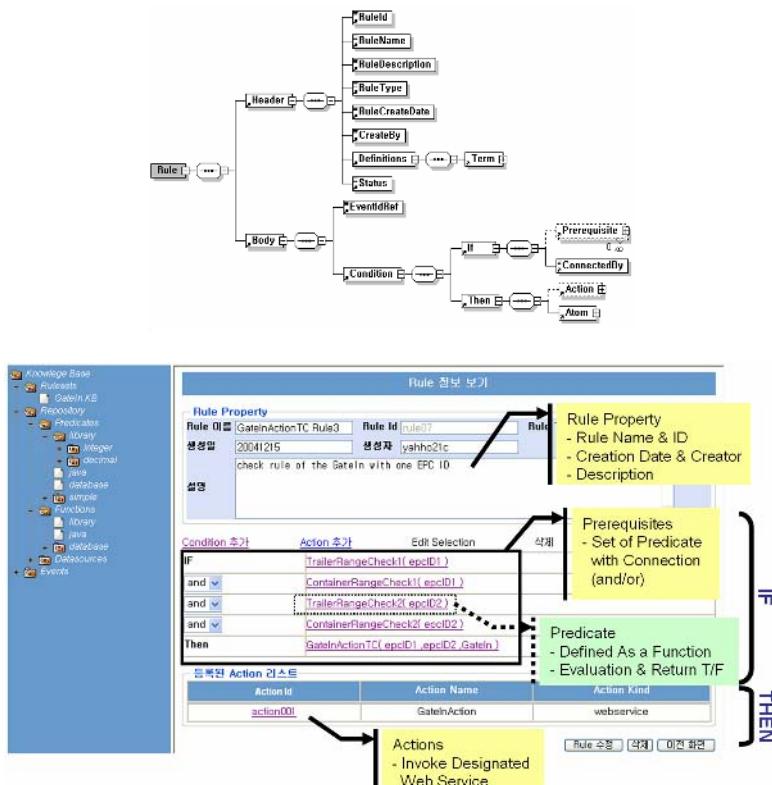


Fig. 9. XML Schema of Rule Definition and Web-based GUI window of Rule Designer. The GUI Window shows a sample rule definition with header, prerequisite and actions. This rule definition instance is presented and stored in the format of XML schema above.

In order to achieve the goal and be suitable for RFID environment, the rule engine of RBPTS adopts the backward chaining inference mechanism. As the physical RFID readers involve the specific business goals – for example, gate open/close, inventory

check and so on – and the business actions triggered by the collected data fall into small number of categories, it is expected that possible conclusions can be chosen at the time that a set of tag data is collected by specified readers. The domain experts define a set of rules which are described as the ‘If condition(s) Then action’ pattern. The event message delivered by REMS includes the ‘action’ indicator called ‘query’ (‘UnloadAction’ of Gate In/Out application in Figure 8) to be proved, so the inference process starts with a conclusion with the help of ‘query’. The rule engine searches for the rule set which have the action clause that matches the action which the event message includes and then evaluate the associated condition clauses. The condition is described as not just a simple form like value matching but also complicated form like a predefined java class or access to database located in the legacy system.

The system architecture of RBPTS and the internal message flow are shown in Figure 8. For the implementation, we revised the open source java class library, MANDARAX [15], in order to implement ECA-based rule engine, and XML Schema for ECA rule definition is newly defined as in Figure 9.

The interaction between the inner components within RBPTS is as follows: REMS accumulates RFID tag data over intervals of time, filters to eliminate duplicate tag data and the tag data that are not of interest, and then reports in the XML/SOAP message form which follows the input format of RBPTS (see Figure 8). The RBPTS-specific event XML messages are generated by custom message dispatcher registered in REMS. Message Receiver accepts the SOAP message and than passes it to upper layer, Event Manager. Event Manager unmarshals the event message, checks whether the message is valid. Afterward, Event Manager reorganize the valid event message into sort of event query message that is used for the next step - inference process - and delivers it to Rule Manager. Rule Manager inquires for the rule set associated with the event query and constitutes all the matters that are essential for the reasoning: database drivers that have access to the legacy database, repository information and so on. Rule Manager feeds all the prepared materials into the Inference Engine and then this evaluates the conditions for each rule and generates the result set. Based on the result set, Rule Manager organizes the action execution list and passes the list to Action Manager. Action Manager searches for the web service for each action execution information and configures the information for the web service call. Action Manager asks for Web Service Agent to call the dynamic web service and records the execution result on the log database. RBPTS supports the application triggering via web service only.

In addition, RBPTS provides web-based user interface for rule design (see Figure 9) to model RFID event, related business rules and the detailed actions which in result provide more flexible way to adapt to the rapidly changing business environment.

Orchestration Engine: Business Process Integration

To construct the simple RFID-based software system, the functionalities provided by RFID Event Management System may be enough; that is, the system collects, filters RFID data coming from tags and simply routes them to enterprise applications. However, we see that the concrete RFID middleware platform must orchestrate RFID-based end-to-end processes that associate with multiple applications or legacy systems and ultimately provide the RFID-related process automation environment.

To be compatible with the idea, we develop the business process engine called ‘Orchestration Engine’ and the system architecture is shown in Figure 10. Currently, the most recent answer to the integration challenge is the Service Oriented Architecture (SOA) and the web service technologies. We suppose that we can access different functionalities of different legacy and other developed applications in a standard way through web services. Under the environment that all applications expose the functionalities via web services, we develop a business process definition and execution engine that provides a way to compose the web service-exposed functionalities in J2EE framework. Mostly, the business processes defined by Orchestration Engine are triggered by RFID-related events including not only the primitive event as the output of data transformation on the RFID IVC but also the semantic event generated by RBPTS.

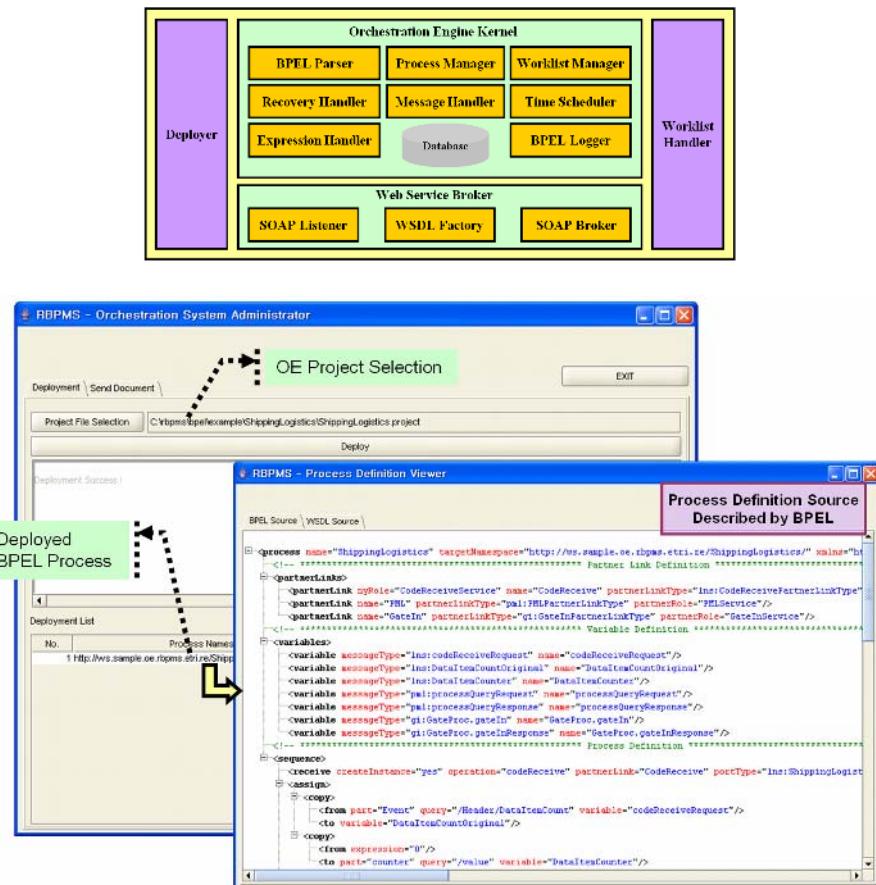


Fig. 10. Architecture of Orchestration Engine (OE) and GUI Window of OE Administrator

In this implementation, we adopt BPEL (Business Process Execution Language for Web Services) [16], an XML-based industry standard for business process

management, as the definition language of business processes. BPEL builds on top of XML and web services, and BPEL process specifies the exact order in which participating web services should be invoked. As the typical scenario we develop under the ETRI RFID Ecosystem, a BPEL business process receives a SOAP request from RBPTS when the raw observation by REMS are dispatched to RBPTS with XML event message and the rule instance, which is invoked by the message and contains the action clause of calling the BPEL process, is evaluated as true. Then, new instance is started and managed by Process Manager. It calls the external web services specified in the BPEL definition – for example, invoking EPCIS web service in order to get the prices of products and then invoking calculator web service to sum up the prices of items which are checked out – and return the results when the process instance is done.

5 Conclusion

RFID technology is known to be well-suited to linking the physical and virtual world and is considered as a key technology to lead us to the ubiquitous computing world.

In this paper, the concept of RFID IVC is introduced and the corresponding activities over the value chain are presented. Furthermore, we demonstrate ETRI RFID Ecosystem that is a RFID middleware platform that materializes the RFID IVC.

The major role of RFID middleware is to collect huge amount of real-time tag read events coming from multiple readers, perform filtering operation on the data stream in order to reduce the data volume, and then share the information about the tagged objects to trading partners and other existing client applications. It is important for the middleware to provide the uniform interface to client applications regardless of heterogeneity among readers from multiple vendors, having their own way of operation and communication.

In order to offer the autonomous system environment starting from the automatic data capture and satisfy the overall RFID IVC, the issue how the more meaningful information or knowledge can be produced through gluing the events captured by RFID readers with legacy systems must be dealt with. In ETRI RFID Ecosystem, we tackle the issue by rule-based semantic event generation and BPEL-applied business process execution triggered by the semantic event.

The prototype system of the introduced middleware software suite is ready to be applied to the field test in order to apply to many business domains such as the gate-in/out management system.

In the next step, our research direction is focused on the fusion of RFID with wireless sensor network. The artifacts we developed mainly deal with ID-based RFID data processing; however, we believe that RFID in conjunction with sensor networks will provide big opportunity and eventually, the RFID middleware platform must be evolved into the ubiquitous middleware platform that can accept all types of AIDC input technologies and handle various types of data. In reality, a lot of activities for combination between RFID and sensors – for example, Smart Active Labels (SAL) [17] – are in progress. As tremendous amount of events which is not only RFID tag data but also sensor data will be generated, the sensor data management will arise as the big problems to be solved. Moreover, context-awareness is distinctive feature for

the case of the integration between RFID and sensors. As an intermediate stage moving toward ubiquitous middleware platform, researches on the integration of the RFID middleware platform with SAL sensor tags and context-awareness are under way.

References

1. Accenture: RFID Execute Overview (2004)
2. EPCglobal Inc.: "<http://www.epcglobalinc.org>"
3. Mealling, M.: EPCglobal Object Name Service (ONS) 1.0 Working Draft, April 15 (2004)
4. Oat Systems and MIT Auto-ID Center: The Savant Version 0.1 Alpha, February (2002)
5. Clark, S., Traub, K., Anarkat, D., Osinski, T.: Auto-ID Savant Specification 1.0 Working Draft, September (2003)
6. Oracle: Oracle RFID and Sensor-Based Services, <http://www.oracle.com/technologies/rfid/index.html>
7. Sun Microsystems: Sun Java System RFID Software, http://www.sun.com/aboutsun/media/presskits/javaone2004/J12004_JavaSystemRFID_Datasheet.pdf
8. Traub, K., Bent, S., Osinski, T., Peretz, S. N., Rehling, S., Rosenthal, S., Tracey, B.: The Application Level Events Specification Version 1.0 Candidate Specification, October (2004)
9. EPCglobal Inc.: EPCTM Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz – 960 MHz Version 1.0.9, January (2005). available from : http://www.epcglobalinc.org/standards_technology/EPCglobalClass-1Generation-2UHFRFIDProtocolV109.pdf
10. ISO/IEC JT1/SC31/WG4: http://usnet03.uc-council.org/sc31/sc31_wg4.cfm
11. Price, J., Jones, E., Kapustein, H., Pappu, R., Pinson, D., Swan, R., Traub, K.: Auto-ID Reader Protocol 1.0 Working Draft, Sep. 5 (2003) 16 ~ 18
12. Alien Technology: <http://www.alientechnology.com/>
13. RFID Solutions from Symbol: <http://www.symbol.com/products/rfid/rfid.html>
14. Floerkemeier, C., Anarkat, D., Osinski, T., Harrison, M.: PML Core Specification 1.0, September (2003). available from : http://www.epcglobalinc.org/standards_technology/Secure/v1.0/PML_Core_Specification_v1.0.pdf
15. The Mandarax Project: <http://mandarax.sourceforge.net/>
16. Business Process Execution Language for Web Services version 1.1: <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>
17. Smart Active Labels Consortium: <http://www.sal-c.org/>

A Collaborative Table Editing Technique Based on Transparent Adaptation

Steven Xia¹, David Sun², Chengzheng Sun³, and David Chen¹

¹ School of Information and Communication Technology,
Griffith University,

Brisbane, Qld 4111, Australia

{Q.Xia, D.Chen}@griffith.edu.au

² Department of Electrical Engineering and Computer Science,
University of California at Berkeley,

Berkeley, CA, USA

DavidSun@berkeley.edu

³ School of Computer Engineering,

Nanyang Technological University,

Singapore 639798

CZSun@ntu.edu.sg

Abstract. Tables are an efficient means to organize information. Collaboration is important for table editing. In this paper, we report an innovative technique, called CoTable, for supporting collaborative table editing in both table-centric and word-centric complex documents. This collaborative table editing technique is based on the Transparent Adaptation approach and hence applicable to commercial off-the-shelf single-user editing applications. Key technical elements of the CoTable technique include: (1) techniques for adapting a variety of table-related data address models, accessible from the single-user Application Programming Interface (API), to that of the underlying Operational Transformation (OT) technique; and (2) techniques for translating user-level table editing operations into the primitive operations supported by OT. The CoTable technique has been implemented in the CoWord system, and CoWord-specific table processing issues and techniques are discussed in detail as well.

1 Introduction

Complex information that includes multiple interrelated items is difficult for human beings to comprehend without a proper organization. Tables are an efficient way to organize such information. A table is usually defined from two perspectives [10]. From the presentation-oriented perspective, a table is a two-dimensional structure consisting of rows, columns and cells. From the structure- or content-oriented perspective, a table is a collection of interrelated information items. Each item is semantically associated with multiple categories. Due to these characters, tables provide a powerful means for facilitating information organization, comprehension, and comparison [16]. In tables, complex information is decomposed into simpler items that are easy to understand. Relationships between items are explicitly labeled so that logical structures of items are presented in a clear form, which facilitates

searching and comparison. Because of the usefulness and convenience, tables are widely used in document processing and generally supported in computer document processing applications such as word processors (e.g. MS Word, OpenOffice Writer), web design systems (e.g. MS FrontPage, Dreamweaver), and spreadsheet systems (e.g. MS Excel, OpenOffice Calc).

In their ethnographic interviews with users of spreadsheets, which are a special form of tables, Nardi and Miller [7] noted that most spreadsheets are developed from collaborative work of users with different expertise. In other words, collaboration is essential for spreadsheet development. Generally, collaboration is also an essential part in table editing.

First of all, the design of tables often involves multiple users. Table authors are aware of what information to present, how the information to be categorized, and what relationships to reveal, but they are not necessarily familiar with techniques for designing table presentational forms that help convey information effectively. Even fewer authors have expertise in using advanced computer table editing features such as formulas and macros. Therefore, table authors often need assistance from table presentation designers with aesthetic, psychological, or computer software knowledge. Although it is the table presentation designer's responsibility to design presentation forms, table authors may also participate in fine tuning table formats (e.g. colors, text fonts, etc.) for better expressing their thoughts. Therefore, collaboration is important in table structure design.

Secondly, filling data into tables also demands multiple users' collaboration. Nowadays, tables are becoming larger and larger in size and more and more complex in structure [10]. As an example, typical spreadsheets for financial or scientific analysis may have thousands of cells. Large tables often include information from different sources. People in charge of these sources are needed to contribute corresponding information to fill the tables. For instance, a financial report table of a bank includes deposit, investment, profit and other information that can only come from different branches and departments. Supporting multiple users to fill tables collaboratively is not only beneficial, but also indispensable in many circumstances.

Collaborative editing techniques and applications have been an active area of research in the field of CSCW, and major progress has been made in the past years [14][17]. However, no adequate attention has been paid to collaborative table editing techniques in previous work. Prior work on collaborative table editing techniques were restricted to table-centric applications like spreadsheets (e.g. Super Spreadsheet [4], and Distributed Spreadsheet [8], but the design of these techniques had not considered their applicability to collaborative table editing in the context of complex documents (e.g. word processor document, HTML document, etc.). Furthermore, these collaborative spreadsheet applications are specially designed for supporting multi-user collaboration (i.e. they are *collaboration-aware* applications [10], which lack adequate conventional editing functionalities available in their commercial off-the-shelf single-user counterparts (e.g. MS Excel and OpenOffice Calc). In this paper, we contribute a novel collaborative table editing technique which can be applied to both *table-centric* and *word-centric* complex document editing applications. More importantly, this technique is based on a novel *Transparent Adaptation* approach [17], which takes advantage of an advanced collaborative technique – *Operational Transformation* [13], and is capable of converting commercial off-the-shelf single-

user editing applications into collaborative ones without changing the source code of the original application.

The rest of this paper is organized as follows. First, the transparent adaptation approach and the operational transformation technique are briefly described to provide the basis for presenting the new work. Next, a novel collaborative table editing technique suitable for both table-centric and word-centric document editing is discussed. Then, this technique is applied to the CoWord system and specific issues in this application are discussed. Afterwards, this work is compared to prior work. Finally, major contributions of this work and some ongoing and future work are summarized.

2 Transparent Adaptation and Operational Transformation

2.1 Basics of the Transparent Adaptation Approach

TA is an innovative approach to converting single-user applications for multi-user real-time collaboration, without changing the source code of the original application [17]. The TA approach is based on a replicated system architecture where the shared single-user application is replicated at all collaborating sites, the use of the single-user application's API (Application Programming Interface) to intercept and replay the user's interactions with the shared application, and the use of the OT technique to manipulate the intercepted user operations for supporting responsive and unconstrained (i.e. concurrent and free) multi-user interactions with the shared application. The central idea of the TA approach is to adapt the data address and operation models of the shared application's API to that of the OT technique.

More precisely, the TA approach can be described by a reference model, as shown in Fig. 1. This reference model consists of three components: *Single-User Application* (SA), *Collaboration Adaptor* (CA), and *Generic Collaboration Engine* (GCE). The main functionalities of these components are sketched below.

The SA component provides conventional single-user interface features and functionalities. This component can be either an existing commercial off-the-shelf single-user application, or a new single-user functionality component in a multi-user collaborative system, but this component itself has no knowledge about multi-user collaboration.

The CA component provides application-specific collaboration capabilities and plays a central role in adapting the SA for collaboration. This component has the knowledge of the SA API but not its internals. At the center of this component is the module of *Adapted Operation* (AO), which represents the SA functionalities exposed by the API. The AO can be generated by the *Local Operation Handler* (LOH) module by intercepting local user's interactions, or received by the *Remote Operation Handler* (ROH) module from remote users. With the AO residing between the API and OT, the task of adaptation between the API and OT is decomposed into two modules:

1. *The API-AO Adaptation* module is responsible for bridging the semantic gap between the API and the AO so that the AO can be correctly replayed on the SA.
2. *The AO-PO Adaptation* module is responsible for mapping between the AO and OT-supported Primitive Operations (PO) so that the underlying OT technique can be used to ensure the correctness of the AO parameters in the presence of concurrency.

The GCE component provides application-independent collaboration capabilities. This component has no knowledge of the single-user application functionality and therefore can be used in adapting different applications. This component encapsulates a package of collaboration supporting techniques, including *Consistency Maintenance* (CM), *Group Undo* (GU), *Workspace Awareness* (WA), and *Session Management* (SM), etc. OT is at the core of this component for supporting consistency maintenance, user-initiated undo, and workspace awareness in a collaborative environment.

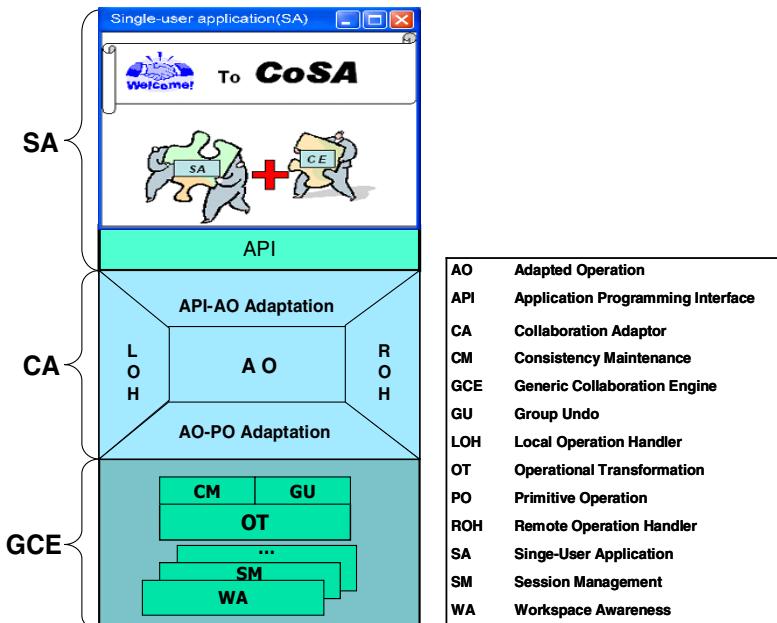


Fig. 1. The TA reference model

2.2 Basics of the Operational Transformation Technique

OT was originally designed to support multiple users to insert and delete characters in replicated text documents concurrently [3][13]. Due to its unique capability in achieving system consistency without imposing any restrictions on users, OT has become the choice of consistency maintenance and group undo technique for many collaborative editing systems [1][5][6][9][14].

The basic idea of OT can be illustrated by using a simple text editing scenario as follows. Given a text document with a string "abc" replicated at two collaborating sites; and two concurrent operations: O1 = Insert [0, "x"] (to insert character "x" at position "0"), and O2 = Delete [3,"c"] (to delete the character "c" at position "3") generated by two users at collaborating sites 1 and 2, respectively. Suppose the two operations are executed in the order of O1 and O2 (at site 1). After executing O1, the document becomes "xabc". To execute O2 after O1, O2 must be transformed against

O1 to become: O2' = Delete [4, "c"], whose positional parameter is incremented by one due to the insertion of one character "x" by O1. Executing O2' on "1abc" shall delete the correct character "c" and the document becomes "xab". However, if O2 is executed without transformation, then it shall incorrectly delete character "b", rather than "c".

In summary, the basic idea of OT is to transform (or adjust) the parameters of an editing operation (e.g. O1) according to the effects of previously executed concurrent operations (e.g. O2) so that the transformed operation (e.g. O2') can achieve the correct effect in the face of concurrency.

There are two underlying models in every OT technique: one is the *data address model* that defines the way data objects in a document are addressed by operations; the other is the *operation model* that defines the set of operations that can be directly transformed by OT functions. Different OT techniques may have different data and operation models. For example, the basic OT technique for plain text editing has an operation model consisting of two POs: *Insert* and *Delete*, and a data address model of a single linear addressing space. Addresses in this linear addressing space ranges from 0 to $N - 1$, where N is the total number of characters in the document.

Despite its text editing origine, OT is independent of text documents and text editing. Particularly, OT does not require the data objects in the document to be of the same type or the same size (though there is only one character type of the same size in plain text documents); OT does not require objects in the document to be presented at the user interface in sequence (though characters in a text document are presented in sequence at the user interface); OT is not restricted to supporting only *Insert* and *Delete* operations (though these two primitive operations are sufficient to support plain text editing); and OT is not restricted to a single linear address model (though this model is adequate for plain text documents). As discovered in prior work [17], OT is applicable to documents consisting of data objects of arbitrary types and sizes, and being presented at the user interface in non-sequential views. Moreover, OT has been extended to support the generic *Update* operation, in addition to *Insert* and *Delete*, for collaborative word processing and to support documents with data objects accessible from a tree of multiple linear addressing domains [2][15].

2.3 Collaborative Table Editing Based on TA and OT

Based on the TA reference model in Fig. 1, a *Collaborative Table* editing technique, named as *CoTable*, is devised and presented in this paper. In this context, the SA component in the reference model can be either a table-centric application like *MS Excel* or a word-centric application like *MS Word*. We assume the SA component has provided the suitable table-related API that meets the basic requirements of the TA approach. The discussion of the CoTable technique shall focus on table-related data address and operation adaptation between the API and the underlying OT technique. Without loosing generality, the discussion of CoTable shall assume the OT technique is based on a data address model with a tree of multiple linear address domains, and an operation model with three POs – *Insert*, *Delete*, and *Update* [2][15].

3 Table Data Address Adaptation

The task of table data address adaptation is to bridge the gap between table-related object data address models exposed by the API and the underlying OT technique. A clear understanding of both ends is an important foundation for designing a proper adaptation strategy.

3.1 Table Data Address Models

When viewed from the *user interface*, a table is a two-dimensional rectangular data structure, consisting of a collection of *rows* and *columns*. Each row or column consists of a sequence of *cells*. A cell may be associated with a row and a column at the same time. A cell may contain some text or graphic objects, which are in a linear sequence. In this *conceptual model* of tables, objects in a table may have various relationships. First, hierarchical relationships exist in the following object pairs: table–column or table-row; column-cell or row–cell; and cell–cell content. Second, objects in the same collection form a separate linear sequence. For example, each cell has an ordinal index in a row, with which the cell can be accessed from the cell sequence in the row. The ordinal indices range from 0 to $N-1$ where N is the number of cells in the row. Removing or inserting cells affects indices of other cells that have higher indices in the same row, but does not affect indices of cells in other rows.

When viewed from the API of an application, the *table data address model* may or may not correspond to the conceptual model. Typically, there are three categories of API table address models, as shown in Fig. 2.

1. *Single linear address model*. In this data address model, table data objects can be accessed from a global linear addressing space of the whole document. Inserting or removing objects contained in cells of a table also affects positions of objects outside the table, and vice versa. Moreover, row/column and cell objects may have marks in the global linear addressing space occupying positions. Taking the global linear addressing space into consideration, this data address model can be represented as a single linear sequence shown in Fig. 2-(a). This data address model can be found in APIs of some word-processing applications including MS Word¹.
2. *Row-based tree address model*. The most significant feature of this model is the absence of columns, and table data objects can be accessed from the row-dimension only. Hierarchical relationships between table-row, row-cell and cell-cell content still exist, and the linear relationships between objects in the same collection remain. This data address model can be represented as a row-based tree shown in Fig. 2-(b). This data address model can be found in APIs of some HTML editors, such as MS FrontPage.
3. *Two-dimensional address model*. In this address model, table data objects can be accessed from both the row-dimension and the column-dimension. This address

¹ API documents for MS Word, MS Excel and MS FrontPage can be found at <http://msdn.microsoft.com>. API documents for OpenOffice Writer and OpenOffice Calc can be found at <http://api.openoffice.org>.

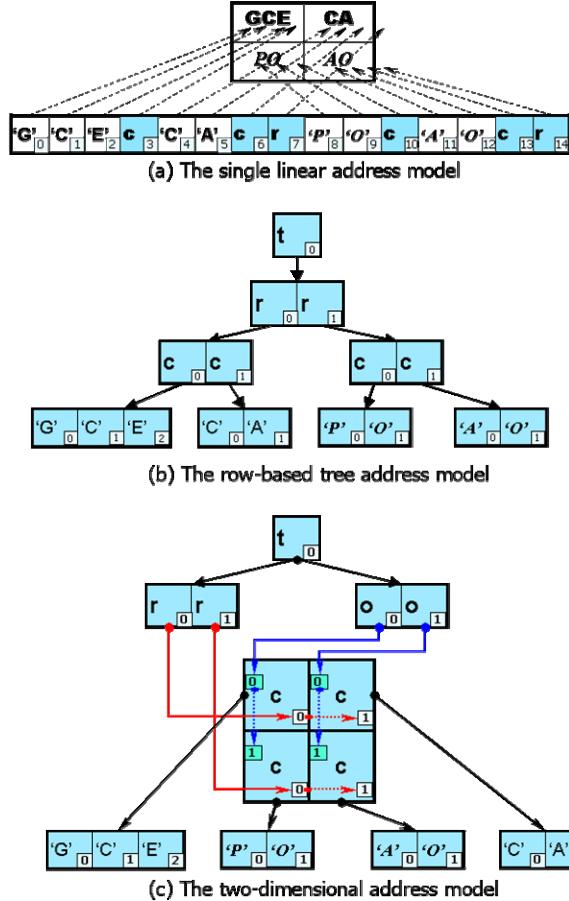


Fig. 2. Table-related data address models in APIs of different single-user applications. In this figure, “t” stands for table; “r” stands for row; “o” stands for column; and “c” stands for cell. The numbers at the lower right corners of each cell stand for object positions in corresponding linear sequences.

model directly matches the conceptual model of tables and can be represented as a hierarchical graph shown in Fig. 2-(c). This data address model can be found in APIs of a variety of single-user applications, including MS Excel, MS PowerPoint, OpenOffice Writer, and OpenOffice Calc.

3.2 The OT Address Model

The *OT address model* defines the way data objects in a document are addressed by operations. In this paper, we assume the OT data address model is a tree of multiple linear address domains [2], as shown in Fig. 3.

In the *OT address model*, a data object is mapped to a position in a linear addressing domain only if it has the position number as its address in this domain. A data object is a *terminal* object if it has no internal data structure or its internal data structure is not addressable. A data object is an *intermediate* object if it has an addressable internal data structure. A terminal object has no link out of it, but an *intermediate* object has a link leading to a lower level addressing domain, which represents this object's internal addressing space.

An object in this data address model can be uniquely addressed by a vector of integers:

$$vp = [p_0, p_1, \dots, p_i, \dots, p_k]$$

where $vp[i] = p_i, 0 \leq i \leq k$, represents one addressing point at level i .

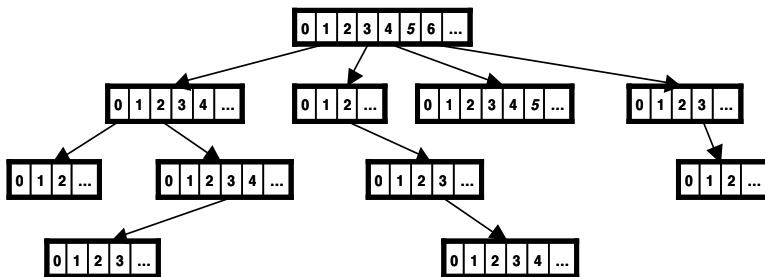


Fig. 3. The *OT address model* used in this paper

3.3 Table Data Address Adaptation Schemes

With the variations of API table data address models, different adaptation schemes are needed. Here we discuss the adaptation schemes for the three API data address models in Fig. 2, respectively.

First of all, the *single linear address model* is a special form of the *OT address model*, in which only the root level domain exists. In the *single linear address model*, an object is uniquely addressed with an integer as the position in the linear sequence. This address is also a special form of the vector address of the OT vector address where $k = 0$.

Moreover, the *row-based tree address model* is also a special form of the *OT address model*, in which (1) the total number of levels is 4, and (2) terminal objects only exist at level 3. In the *row-based tree address model*, an object is uniquely addressed with a vector of integer. This address is also a special form of the OT vector address where $k \leq 3$.

Finally, the *two-dimensional address model* is not directly compatible with the *OT address model* due to the dual hierarchical relationships between cells and rows/columns. However, a comparison of the *two-dimensional address model* and the *row-based tree address model* reveals that removal of column objects from the *two-dimensional address model* reduces the dual hierarchical relationships to a single one

and hence converts the *two-dimensional address model* to the *row-based tree address model*.

In summary, the three API data address models are all adaptable to the *OT address model*. The *single linear address model* and the *row-based tree address model* are adapted directly; and the *two-dimensional address model* is adapted after a conversion to the *row-based tree address model*.

3.4 Discussions

There are three issues worth discussing in the data address adaptation schemes. First, it is theoretically equivalent to remove either columns or rows in the adaptation of the *two-dimensional address model*, because both a row-based and a column-based tree can be adapted to the *OT address model*. Without loosing generality, the following discussions shall be based on the assumption of a row-based tree.

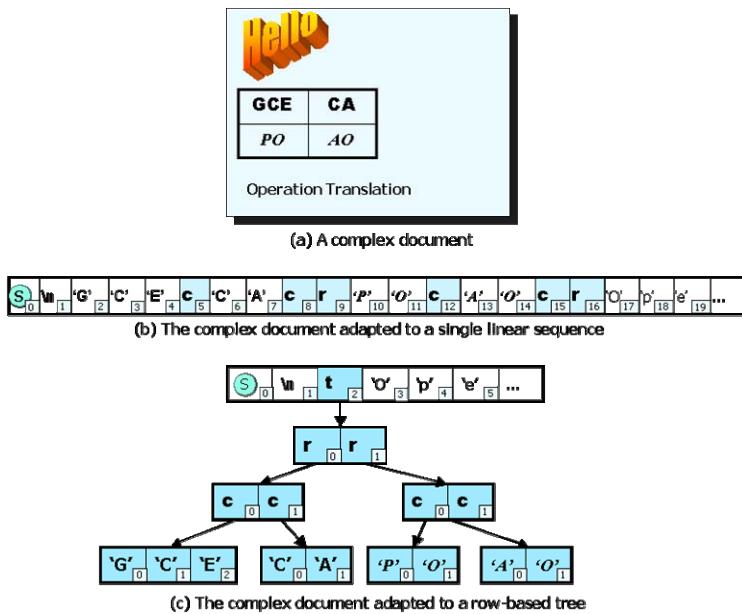


Fig. 4. Integrating the table into the global addressing space of the complex document

Second, the row-based tree converted from the *two-dimensional address model* does not need to be semantically equivalent to its original two-dimensional form. The conversion process selectively preserves some information about the table structure but discards some others, including the hierarchical relationships between cells and columns. This is acceptable because the *OT address model* only needs to maintain information relevant to OT. For example, OT only needs to know the vector position of a cell in the *OT address model*, regardless which column the cell is subordinate to, so information about columns can be ignored. However, it is important for OT to

know that one cell is located before another in the same collection, so such information is retained.

Third, the adaptation schemes not only provide a solution to mapping the API table data address model into that of OT, but also is the key technique to integrate tables into the global addressing space of the complex document, as shown in Fig. 4. The complex document in Fig. 4-(a) includes three parts. The first line contains an inline graphic object “Hello”, followed by a *Return* character. Afterwards there is a table containing two columns and two rows. The last line contains some text. According to the data address adaptation technique in [17], the graphic object and the text segment can be mapped into two linear sequences separated by the table. Based on different data address models exposed by the API, the table can be adapted to a single linear sequence or a row-based tree. Both adapted models can be merged with the linear sequence of objects outside the table. The merged data models of both cases are shown in Fig. 4-(b) and (c), respectively. It is clear that both merged models are compatible with the *OT address model*.

4 Table Operation Adaptation

Table-related AOs could target on objects contained in table cells (e.g. text or graphics) or table structure objects (e.g. cell or row). The data address adaptation schemes have integrated objects in a table into the global OT addressing space for the whole document, so AOs used to manipulate objects (e.g. text or graphics) outside a table can also be used for objects inside a table, and the operation adaptation techniques for existing AOs can be directly inherited. However, table structure operations are table-specific and cannot be supported by existing AOs designed for graphics or text [17]. They require specific adaptation techniques. Therefore, our discussion on CoTable operation adaptation shall focus on the issues and techniques related to table editing operations only.

4.1 Operation Models of the API and OT

Table operation adaptation is responsible for the translation between the table-editing API and POs supported by OT. Before designing adaptation approaches, operation models of the API and OT are discussed in this subsection.

The API table operation model defines the set of operations that could be performed to and have effects on table structure objects. There are three factors contributing to the determination of these operations’ effects, i.e. the API operation type, the target object type, and the underlying data address model.

First, generally the table-editing API have three operation types, including (1) *inserting* new objects, (2) *deleting* existing objects, and (3) *updating* attributes of existing objects. The first two determine the existence of objects and the last one changes object states. Second, the target object type determines the effect scope. For example, creating a cell causes one object to be inserted into the data address model, but creating a row involves multiple objects. Third, the target object type and the data address model have combined effects on the topology of the effect ranges. For example, the effect range of creating a row is a sub-tree in a *row-based tree address*

model, but is a continuous sequence in a *single linear address model*. Moreover, the effect range of inserting a column includes dispersed objects in both data address models.

In the OT operation model, three generic POs are defined on its data address model:

1. *Insert* PO inserts a sequence of new objects into the *OT address model*.
2. *Delete* PO removes a sequence of existing objects from the *OT address model*.
3. *Update* PO changes attributes of a sequence of existing objects in the *OT address model*.

These POs are generic in the sense that they are independent of object types. With these POs, OT does not need any application-specific knowledge to do its work.

4.2 AOt Definition

The solution to bridging the gap between these two operation models is to define a set of table structure AOs, denoted as AOt. As a vehicle for the translation between the API and POs, the AOt should (1) correctly reflect API's effects by covering all affecting factors, and (2) facilitate translation between the API and PO.

Consequently, AOt are organized in two dimensions. One dimension is based on the types of table structure object that the AOt targets. There exist three table structure object types: *row*, *column* and *cell*. Therefore, we have three AOt categories in this dimension, which are *Row-AOt*, *Column-AOt* and *Cell-AOt*. This dimension not only facilitates the translation between the API and the AOt, but also indicates the object type's influence on API effects.

Another dimension is based on the API operation types. Because the three API operation types exactly correspond to the three PO types, this dimension facilitates the translation between the AOt and POs. The three AOt categories in this dimension include *Insert-AOt*, *Delete-AOt*, and *Update-AOt*.

Based on this two-dimensional classification, any AOt can be placed in a suitable cell in Table 1. In fact, there are many more AOt in real applications than what are listed in Table 1. For example, additional *Cell-Update-AOt* may include *Change_CellFillColor*, *Change_CellBorderStyle*, *Change_CellBorderColor*, etc. Nevertheless, for the purpose of investigating issues of operation translation, the AOt listed in Table 1 are representative and adequate.

Table 1. AOt classification

<i>PO</i> \ <i>Obj</i>	Row	Column	Cell
Insert	Ins_Row (vp, len, row)	Ins_Col (listof<vp, len>, col)	Ins_Cell (vp, len, cell)
Delete	Del_Row (vp, len, row)	Del_Col (listof<vp, len>, col)	Del_Cell (vp, len, cell)
Update	Change_rowHeight (vp, len, o_val, n_val)	Change_ColWidth (listof<vp, len>, o_val, n_val)	Change_cellColor (vp, len, o_val, n_val)

Parameters of the AOt show that they are defined directly on the *OT address model*. The parameter *vp* is a vector of integer. It indicates the starting position of an AOt effect range in the *OT address model*. The parameter *len* indicates the length of an AOt effect range. Apart from positional references, we also keep other parameters that are needed in OT. For *Insert-* and *Delete-AOt*, we keep the objects affected by the AOt as the last parameter: *row*, *col* or *cell*. For *Update-AOt*, we record the old value *o_val* and new value *n_val* of the target attributes. These parameters are needed in OT for consistency maintenance and group undo [11][15].

The effect range parameters (*vp* and *len*) are able to locate any continuous range in the *OT address model*, which is able to accommodate both the *based tree* and the *single linear address models*. Therefore, for an AOt that has a single continuous effect range (*Row-* or *Cell-AOt*), the effect range parameters are sufficient in any API data address models. Nonetheless, a *Column-AOt* has dispersed effect ranges in both API data address models because of the dispersed distribution of cells in a column, so a list of range parameters is needed.

With the AOt definition in Table 1, the translation from the AOt to both PO and the API are straightforward. On the one hand, in AOt-PO translation, the PO type is just the PO category of the AOt; the PO effect range parameters are the same as that of the AOt. On the other hand, in API-AOt translation, the effect range parameters are used to locate the target object in the API addressing space; the target object type encoded in the AOt type provides information about the target object's API interface (e.g. method definitions); the PO type encoded in the AOt type is used to choose the method to invoke; other AOt parameters are used as method invocation parameters.

5 Supporting Collaborative Table Editing in CoWord

CoWord is a TA-based application which converts MS Word into an unconstrained collaborative word processor without changing the source code of Word. To support collaborative table editing in CoWord, we have implemented the CoTable technique in the CoWord system. Application-specific issues emerged in adapting Word API address and operation models are discussed in this section.

5.1 Special Issues in Word Table Data Address Adaptation

The Word API exposes a *single linear address model*. Objects inside table cells and outside ones can be accessed with their positional references that are defined in a global linear addressing space. Moreover, there are *end-of-cell* and *end-of-row* marks for each cell and row in this global linear addressing space with unique positions. Therefore, all objects in a Word document can be mapped to proper positions in the *OT address model* as shown in Fig. 4-(b).

However, some objects in a Word document are hidden in both the user interface and the API. To ensure the correctness of the data address adaptation, it is important that these objects also be located and mapped to the *OT address model*. One example of such hidden objects is the invisible cells generated while handling irregular tables.

Some Word tables are irregular, in the sense that some cells cannot be definitely subordinated to certain rows or columns. Fig. 5-(a) and (c) show tables that are irregular in two different dimensions.

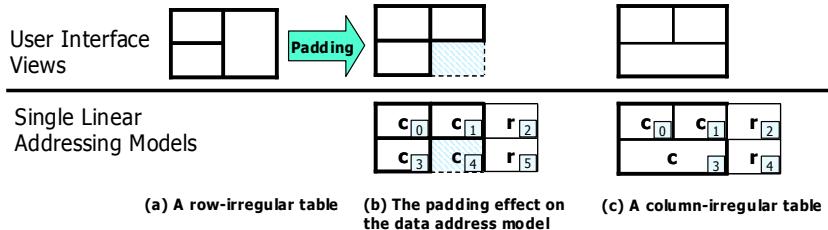


Fig. 5. Handling irregular tables and its effects on the data address model. To match the user interface views of tables better, single linear address models are shown in rectangular forms.

In the row-irregular table in Fig. 5-(a), an ambiguity exists in determining which row the right cell belongs to, because it spans two rows. In the Word API data address model, this cell is associated with the upper row. At the same time, an invisible cell is padded beneath the spanning cell in the lower row to eliminate the ambiguity (shown in Fig. 5-(b)). In contrast, in the column-irregular table in Fig. 5-(c), no padding is performed.

Such invisible cells must not be ignored in the data address adaptation. Although these cells are invisible in the user interface and inaccessible from the Word API, they are also assigned with positions in the global linear addressing space. Ignoring these cells would have the consequence of ruining the correctness of the data address adaptation.

5.2 Special Issues in Word Table Operation Adaptation

Understanding effects of Word table-editing functionalities for the AOt generation. As a TA-based system, CoWord generates AOt by intercepting the user's interaction with the Word user interface; the user's interactions may trigger Word table-editing functionalities to change the document state. Therefore, an important basis for the AOt generation is a precise understanding of these functionalities' effects.

Word table-editing functionalities have visible effects on the user interface and invisible effects on the API data address model. In most cases, these effects are consistent, but sometimes may be inconsistent. Under any circumstances, the generation of AOt should always be based on the API data address model effects.

One example where this inconsistency occurs is the vertical cell merge, whose effects on the user interface and the data address model are shown in Fig. 6. When two cells are merged vertically, the effects on the user interface is that the lower cell is removed and the upper cell spans two rows. This vertical merge causes irregularity, so the padding scheme is applied (by Word) in the data address model. As shown in Fig. 6, there is no positional difference between the data address model states before

the merge and after the padding. The only difference is that the lower cell becomes invisible. According to this data address model effect, a *Cell-Update-AOt* needs to be generated to set the *visibility* attribute of the lower cell to false.

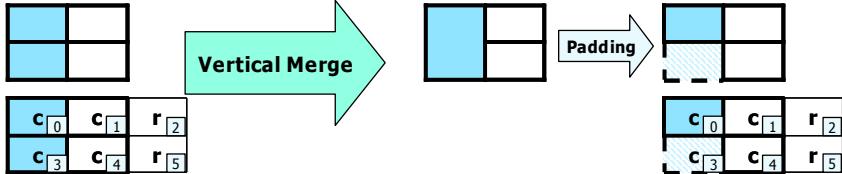


Fig. 6. Effects of vertical cell merge on the user interface and data address model

Preserving regularity effects of the AOt in the face of concurrency. TA-based systems allow users to use the single-user functionalities and interface features in the same way as in the single-user environment. So, it is natural for users to expect the same effect of these conventional functionalities in the collaborative environment, thus it is desirable to require the underlying system to preserve the effect of single-user functionalities in the collaborative environment.

In the single-user environment, only *Ins_Cell* and *Del_Cell* AOt could irregularize a regular table; the application of a *Row/Column-AOt* to a regular table preserves the regularity of the table. This regularity effect of the AOt should be preserved in the collaborative environment. However, in the *single linear address model* of the Word API and in the face of concurrency, the regularity effect may be lost without a special treatment.

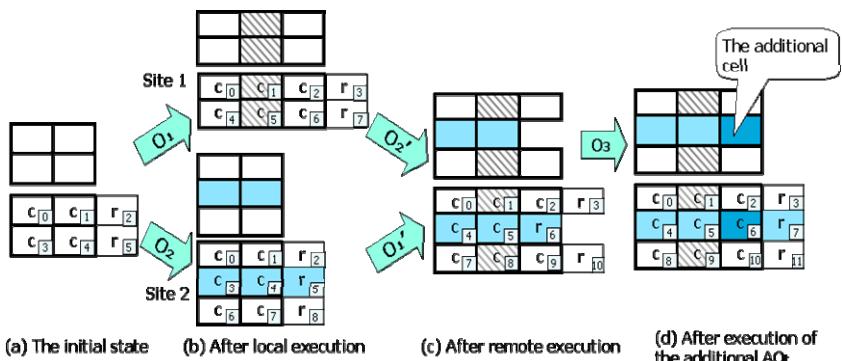


Fig. 7. Preserving the regularity effects of *Ins_Row* and *Ins_Col* AOt. In this figure, O1=Ins_Col (<1, 1>, <5, 1>, col); O2=Ins_Row (3, 3, row); O3=Ins_Cell (6, 1, cell); O1' and O2' are OT-processed forms of O1 and O2.

As shown in Fig. 7, from the initial table state (shown in Fig. 7-(a)), site 1 generates an AOt O1 that inserts a new column. Concurrently site 2 generates an AOt O2 that inserts a new row. Both *Insert-AOt* contain two cells. After executed locally

(shown in Fig. 7-(b)), they are propagated to remote sites. When O1 arrives at site 2, it is translated to POs, processed by OT and executed, which results in the insertion of two cells and leads to the table state shown in Fig. 7-(c). Site 1 goes through a similar process after the arrival of O2 and reaches the same table state.

The table in Fig. 7-(c) is an irregular one, whose irregularity comes from the combined effect of two concurrent *Row*- and *Column*-AOt. In other words, the regularity effect of these two AOt is lost in the face of concurrency.

The correct combined result of these two AOt should be that shown in Fig. 7-(d), where the regularity is still preserved after the insertion of a row and column. The difference between the tables in Fig. 7-(c) and (d) is that the one in Fig. 7-(d) has an additional cell, which helps preserve the table's regularity. To convert the table state from that shown in Fig. 7-(c) to (d), an additional *Ins_Cell* operation O3 is needed to insert that additional cell.

A thorough investigation shows that this problem occurs only when column AOt (i.e. *Ins_Col*, *Del_Col* and *Upd_Col*) are concurrent with the *Ins_Row* AOt and their target ranges are in the same table. An additional AOt needs to be generated in these cases to preserve the table's regularity.

6 Comparison to Prior Work

Prior work on collaborative table editing has been restricted to collaboration-aware table-centric (spreadsheet) applications. The CoTable technique is unique in providing a collaborative table editing solution to both table-centric and word-centric applications and in its applicability to commercial off-the-shelf single-user editing applications without changing the source code of the original application.

Super Spreadsheet [4] is a collaborative spreadsheet system for face-to-face users. Management of concurrency, spreadsheet version and history is performed in an object-oriented way. For concurrency control, a transaction-based approach was adopted. The user's interactions with the system are organized into transactions. During the execution of a transaction, implicit locks are used to lock the data objects before updating (i.e. pessimistic locking), and locks are released at the transaction commitment time, which is chosen by the user explicitly. Locks of multiple objects can be acquired in arbitrary orders (i.e. non-strict 2-phase locking), so deadlock is possible. There exist automatic deadlock detection mechanisms in the system but users must be involved in deadlock resolution by negotiation. This solution works well in the local-area network environment (for face-to-face users). However, if this approach were applied in the Internet environment, the system responsiveness may suffer due to the use of pessimistic locks.

Similarly, the Shared Spreadsheet in WARP project² also takes a transaction-based approach as its concurrency control mechanism. Users need to explicitly start a transaction before editing and end the transaction afterwards. Transactions failing in conflicts have to be rolled back, which may result to the loss of collaborative work. Besides, a series of auxiliary features are implemented to increase performance and reduce the possibility of rolling back.

² WARP: Wide Area Resource Programme. <http://distsyst.dcs.st-and.ac.uk/warp/warp.html>.

Transaction/lock-based concurrency control solutions are able to protect data integrity by prohibiting conflicting updates on shared data objects, which is important in achieving *semantic consistency* in collaborative applications. On the other hand, OT-based solutions can ensure *syntactic consistency* (characterized by *convergence*, *intention-preservation*, and *causality-preservation* [14]), and provide high responsiveness, fine-grain concurrency, and high degree of freedom to the users in their interactions with the shared application in the Internet environment. In our view, OT and transaction/locking are complementary techniques and could be integrated for achieving both syntactic and semantic consistency. In [12], an optional and responsive fine-grain locking scheme has been devised for Internet-based collaborative systems. We are in the process of integrating the locking scheme in [12] into the GCE component of the TA approach.

An OT-based distributed collaborative spreadsheet system was proposed by Palmer and Cormack [8]. Their OT technique is specially designed for supporting the *two-dimensional address model*, and spreadsheet-specific operations, which are *insert* and *delete* rows or columns in a table, and *set*, *format*, and *copy* the cell value of a table. These spreadsheet-specific operations are at the same level as the AOt in our approach. In contrast, our approach is based on an OT technique which directly support only three generic primitive operations (*Insert*, *Delete*, and *Update*), and an adaptation technique to map application-level table operations (i.e. the AOt) into these primitive operations. The benefit of our approach is the reduced complexity in designing transformation functions and the reusability of transformation functions. Moreover, our approach has the unique advantages in providing a collaborative table editing solution to both table-centric and word-centric applications and in its applicability to commercial off-the-shelf editing applications.

7 Conclusion and Future Work

In this paper, we have contributed an innovative technique CoTable to supporting collaborative table editing in both table-centric and word-centric complex documents. The CoTable technique is based on the Transparent Adaptation (TA) approach, and therefore applicable to a range of existing commercial off-the-shelf single-user editing applications. The key elements of the CoTable technique include the techniques to adapt the table-related data address and operation models to that of the underlying Operational Transformation (OT) technique. Because of the OT support, the CoTable technique is able to support unconstrained (i.e. concurrent and free) editing of any table structure and content elements. Moreover, the proposed CoTable technique is compatible to existing data address and operation adaptation techniques for other data types (e.g. text and graphics) in complex documents.

The CoTable technique has been applied to the CoWord system to test its correctness and feasibility. This work has also enriched our knowledge and experience in designing TA-based collaborative techniques and applications, and provided new evidence on the power and generality of the TA approach.

We are in the process of applying the CoTable technique to other single-user editing applications, including Excel and FrontPage. Moreover, we plan to investigate issues in applying the TA approach to CAD/CASE tools.

References

1. Begole, J., Rosson, M., and Shaffer, C.: "Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems", *ACM Transactions on Computer Human Interaction*, vol. 6, no. 2, 1999, pp. 95 – 132.
2. Davis, A., Sun, C., and J. Lu.: "Generalizing operational transformation to the standard general markup language", *In Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, November 2002, pp. 58 – 67.
3. Ellis, C. A., and Gibbs, S. J.: "Concurrency control in groupware systems", *In Proceedings of ACM Conference on Management of Data*, May 1989, pp. 399 – 407.
4. Fuller, D. A., Mujica, S. T., and Pino, J. A.: "The design of an object-oriented collaborative spreadsheet with version control and history management", *In Proceedings of the ACM/SIGAPP symposium on Applied computing: states of the art and practice*, 1993, pp. 416 – 423.
5. Ignat, C., and Norriei, M.C.: "Customizable collaborative editor relying on treeOPT algorithm", *In Proceedings of the European Conference of Computer-supported Cooperative Work*, September 2003, pp. 315 – 324.
6. Li, D. and Li, R.: "Transparent sharing and interoperation of heterogeneous single-user applications", *In Proceedings of ACM Conference on Computer Supported Cooperative Work*, November 2002, pp. 246 – 255.
7. Nardi, B. A. and Miller, J. R.: "An ethnographic study of distributed problem solving in spreadsheet development", *In Proceedings of the ACM conference on Computer-supported cooperative work*, 1990, pp. 197 – 208.
8. Palmer, C. R. and Cormack, G. V.: "Operation transforms for a distributed shared spreadsheet", *In Proceedings of ACM Conference on Computer Supported Cooperative Work*, November 1998, pp. 69 – 78.
9. Ressel, M., Nitsche-Ruhland, D and Gunzenbauser, R.: "An integrating, transformation-oriented approach to concurrency control and undo in group editors", *In Proceedings of ACM Conference on Computer Supported Cooperative Work*, November 1996, pp. 288 – 297.
10. Silberhorn, H.: "TabulaMagica – an integrated approach to manage complex tables", *In Proceedings of the ACM Symposium on Document engineering*, 2001, pp. 68 – 75.
11. Sun, C.: "Undo as concurrent inverse in group editors", *ACM Transactions on Computer-Human Interaction*, vol. 9, no. 4, December 2002, pp. 309 – 361.
12. Sun, C.: "Optional and responsive fine-grain locking in Internet-based collaborative systems", *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 9, September 2002, pp. 994 – 1008.
13. Sun, C. and Ellis, C. A.: "Operational transformation in real-time group editors: issues, algorithms, and achievements", *In Proceedings of the ACM Conference on Computer Supported Cooperative Work*, 1998, pp. 59 – 68.
14. Sun, C., Jia, X., Zhang, Y., Yang, Y., and Chen, D.: "Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems", *ACM Transactions on Computer-Human Interaction*, vol. 5, no. 1, March 1998, pp. 63 – 108.
15. Sun, D., Xia, S., Sun, C., and Chen, D.: "Operational transformation for collaborative word processing", *In Proceedings of ACM Conference on Computer Supported Cooperative Work*, November 2004, pp. 437 – 446.
16. Wang, X.: "Tabular abstraction, editing, and formatting", PhD thesis, University of Waterloo, Ontario, Canada.
17. Xia, S., Sun, C., Sun, D., Shen, H., and Chen, D.: "Leveraging single-user applications for multi-user collaboration: the coword approach", *In Proceedings of ACM Conference on Computer Supported Cooperative Work*, November 2004, pp. 162 – 171.

Inter-enterprise Collaboration Management in Dynamic Business Networks

Lea Kutvonen, Janne Metso, and Toni Ruokolainen

Department of Computer Science, University of Helsinki, Finland
{Lea.Kutvonen, Janne.Metso, Toni.Ruokolainen}@cs.Helsinki.FI

Abstract. The agility to collaborate in several business networks has become essential for the success of enterprises. The dynamic nature of collaborations and the autonomy of enterprises creates new challenges for the operational computing environment. This paper describes the web-Pilarcos B2B middleware solutions for managing the life-cycle of dynamic business networks in an inter-enterprise environment. The use of B2B middleware moves the management challenges away from the individual enterprise applications to more global infrastructure services, and provides a level of automation into the establishment and maintenance. The middleware services aim for a rigorous level of transparent interoperability support, including awareness of collaboration processes, and collaboration level adaptation to breaches in operation.

Keywords: E-services architectures and technologi; Inter-enterprise eCommunity management, interoperability; Corss-organisational process support, contracts.

1 Introduction

The present, rapid globalization of business makes enterprises increasingly dependent on their cooperation partners; competition takes place between supply chains and networks of enterprises. The level of dynamic integration capabilities between independent enterprise ICT systems is critical for the success of such business networks. Enterprise ICT systems are expected to participate into several, potentially heterogeneous networks simultaneously. They should also be able to react fast to changing partnerships, and use technology-independent tools for managing technical and semantical interoperability.

Traditional inter-enterprise integration solutions are typically based on tightly coupled application level integration (EAI) or they rely on some common meta-model to generate interoperable business applications. The use of these integrated or unified collaboration models usually guarantees the correct operation of inter-enterprise communities as all the needed interoperability information is implicitly contained in the resulting inter-enterprise applications. However, interoperability is achieved at the expense of autonomy, reusability and flexibility of business services and networks.

Possibilities for service reuse and evolution, as well as business network adaptivity, can be enhanced by the use of a federated collaboration model. The federated model builds collaboration relationships between already existing services, based on their interoperable functionalities.

The federated approach needs a platform with facilities for inter-enterprise network management and for making the interoperability information explicitly available during community operation. A strategic breeding environment for new business networks is needed with facilities for deciding on the shared business process, and roles of partners within it; selection of component services from the partners' IT systems; ensuring and enforcing interoperability between the component services; and establishing the business network. An operational environment for maintaining and controlling the business network is needed with facilities for joining and leaving the network; automated monitoring of the behaviour of the network and intelligent methods for adapting to technological changes and heterogeneity in the processing environment; and adapting to collaboration changes in terms of network membership and breach management.

The web-Pilarcos project aims for a decrease in the cost of establishing and operating electronic business networks, especially in the cost involved into changes of the business processes, partnerships, application services, and platform technologies. The main investment must be placed on the right kind of middleware that is able to use metainformation on the changeable elements for governing the overall collaborations. As the web-Pilarcos middleware is directed for enterprises participating in multiple, heterogeneous business networks where gradual evolution is to be expected, we call it B2B middleware. It forms a loosely-coupled collaboration layer on top of distributed, service oriented middleware.

The web-Pilarcos architecture uses meta-level information – such as business process model and service descriptions of the participants – that via reflection mechanisms governs the business network operation. The meta-level information can be renegotiated and changed, and these contractual changes are automatically reflected to the underlying computing system configuration. Likewise, automated mechanisms are build to observe the underlying system status, and reflecting that back to the status of the meta-information.

The web-Pilarcos architecture represents an approaches where meta-level contracts are used for inter-enterprise collaboration management. In contrast to most other architectures, web-Pilarcos does not use the metamodels for executing the collaborative workflow, but to check the potential for process-level and pragmatic interoperability and to monitor conformance to the agreed collaboration model. When interoperability is achievable, the collaboration establishment phase is able to automatically configure some adaptors to the runtime environment; when operational-time breaches are detected, resolution processes can be automatically initiated across the collaboration. This approach is more cost-effective in terms of tolerating changes in local and collaborative business processes, provided services, and platforms.

This paper describes the web-Pilarcos B2B middleware solutions for managing the life-cycle of dynamic business networks in an inter-enterprise envi-

ronment. Section 2 introduces the B2B middleware services and eCommunity contracts. While Section 3 briefly addresses the role of breeding environment, Section 4 elaborates on the operational time services. Section 5 discusses the prototype implementation. Related work and future development issues conclude.

2 The B2B Middleware Services

To facilitate joint management of collaborations in the web-Pilarcos architecture, inter-enterprise collaborations are modelled as eCommunities that comprise of independently developed business services. The inter-enterprise collaboration management is supported by concepts of

- an eCommunity that represents a specific collaboration, its operation, agreements and state; the eCommunities carry identities and are managed according to their eCommunity contract information; and
- services that are provided by enterprises, used as members in eCommunities, and are made publicly available by exporting service offers.

These concepts are used by a set of B2B middleware services for establishing, modifying, monitoring, and terminating eCommunities. Looking from the application service point of view, operations are made available for joining and leaving an eCommunity either voluntarily or by community decision.

For the eCommunity management, interoperability is a fundamental issue. Interoperability, or capability to collaborate, means effective capability of mutual communication of information, proposals and commitments, requests and results. Interoperability covers technical, semantic, and pragmatic interoperability. Technical interoperability means that messages can be transported from one participant to another. Semantic interoperability means that the message content becomes understood in the same way by the senders and the receivers. This may require transformations of information representation or messaging sequences. Finally, pragmatic interoperability captures the willingness of partners for the actions necessary for the collaboration. The willingness to participate involves both capability of performing a requested action, and policies dictating whether the potential action is preferable for the enterprise to be involved in. In the pragmatic view, process-awareness in terms of collaborative business process model is needed, augmented with nonfunctional aspects, some of which are related to business policies.

The interoperability challenges are addressed both by the breeding facilities and operational environment. The breeding environment supports establishment of eCommunities in such a way that the open markets of business services is exploited, but at the same time the strategic and pragmatic restrictions of involved enterprises are taken into consideration. The operational environment is responsible of observing the behaviour of the participants in the eCommunity, react to breaches of the eCommunity contract, and to respond to eCommunity administrator or participant requests (human intervention) on renegotiation of some contract aspect. The breeding and operational environments are not isolated

from each other, as for example joining new members to an existing eCommunity involves services of the breeding environment. The metainformation services and their use for interoperability ensurance is briefly summarized in Section 3 (a more thorough discussion is given in [1, 2]).

- reference to the business network model;
- information about the epoch in which the network is;
- process for changing epoch;
- for each role
 - assignment rules that specify the requirements on
 - * service type;
 - * nonfunctional aspects;
 - * restrictions on identity, participation on other business networks, etc;
 - conformance rules that are used for determining conformance to the role which the assigned component is in the role; similar as above;
- for each interaction relationship between roles
 - channel requirements
 - locations of the channel endpoints
 - QoS agreement; security agreement
 - information presentation formats
- for each policy that governs the choices between alternative behaviour patterns in the business network model
 - acceptable values or value ranges;
- references to alternative breach recovery processes;
- objective of the business network as business rules

Fig. 1. Information contents of the eContract

The operational environment supports the metalevel model of eCommunity by maintaining (distributed) eContracts. The semantical contents of the eContracts is summarized in Fig. 1. The eContract captures information about the agreed business network model, the participants with their locations and service access points, rules for accepting partners into the network, rules for monitoring whether existing partners can be accepted to continue in the network, and agreed collaborative process models for breach recovery situations. For each communication relationship, the eContract also captures the requirements for the abstract communication channel needed; this channel is then further mapped to suitable distribution platform services. The eContracts can be changed by negotiation between agents representing partners (enterprise) of the eCommunity. These agents do not themselves provide the involved services, but reside at the B2B middleware level, and act based on business-rules and process-models defined for the application service in question and notifications reporting the progress and the failures of the collaboration in question. The agents can control the local operating environment through the local service management facilities. For interoperability, the shared metalevel notations in the eContract are transformed to the locally understood management data and methods.

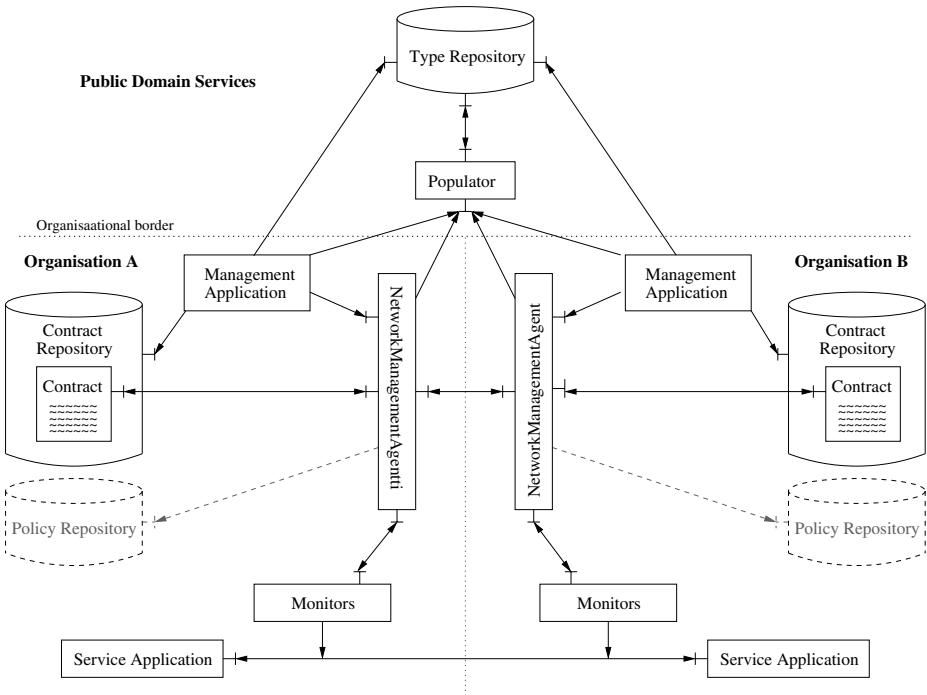


Fig. 2. Service agents of the operational environment

Fig. 2 illustrates the B2B middleware service agents of the operational environment. Each site or administrative domain, representing an autonomous ICT system, is expected to run a business process management agent. By autonomy we mean the potential for control over the private computing systems, and moreover on strategic business processes and policies. Breeding environment services like populators and type repositories are not required from all sites, but can be provided as infrastructure services as a business on its own right.

3 Breeding Environment

Establishment and maintenance of eCommunities relies on the interoperability knowledge on business network models (BNM), service types and associated information, and service offers.

The business network models are defined in terms of roles and interactions between the roles. For each role, assignment rules define additional requirements for the service offer that can be accepted to fulfill it, and conformance rules determine limits for acceptable behaviour during the eCommunity operation. Thus the business network model defines the structure and behaviour of the collaborating community. A verified business network model acts as a template for the eCommunity.

The model to be used as a contract template is first negotiated between the potential partners, involving comparison and matching of strategical, pragmatical goals of members in the network. The matching of network models is too hard a problem to solve by an automated process in general cases for a heterogeneous modeling environment. Therefore, we have focused on practical goals: What is needed is a grouping of similar models, where there is suitable transformers or adapters available for configuring a communication channel between peers so that the information exchange becomes understood correctly and there is no known deadlock in the sequence of message exchanges. The adapters can address modifications at multiple levels of interoperability, such as data representation modifications, and changing the communication pattern (for example, splitting a request of a task to a set of requests for subtasks from the peer). The service type repository is used for holding such relationships between models and the transformation information associated. The actual adapters are produced in a separate process starting from the service type descriptions [3].

Potential participants for the defined roles are retrieved from the service offer repository based on their service type and published properties that are associated for that type. Service type information captures syntactic and procedural interoperability; semantics over behaviour is considered too hard yet [4]. On one hand, the business network models associate roles with required service types; on the other hand, the service offer repository associates the service with the service types. The quality of these assertions is essential for the correctness of the created communities. Furthermore, the density of the created relationship network of alternative but interoperable service types determines the usefulness of the middleware service [5].

The resulting eCommunity contract object is an active agent itself, and provides a service interface with operations for initiating re-negotiations, and receiving progress reports by participants. It also takes initiative in message exchanges with local service management agents at each site involved.

The repositories that maintain a growing and increasingly interrelated set of interoperability knowledge are feed by independent processes: publication of a) service types and b) business network models, c) service offers, and d) eCommunity population and negotiation processes. These processes are inter-related but not tightly dependent; for example new service types can be published without a business network model using them. Fig. 3 illustrates these processes. The service publication functionality is similar to the UDDI [6] or the ODP trading mechanisms [7]; while the type management system resembles the ODP type repository function [8] and enforces a typing discipline to follow over service offer repositories. The BNM repository is a shared storage of business collaboration information that enable enterprises to share business transaction models, such as the ebXML-repository [9], although with more automated and repeatable breeding process. The used notations are not discussed here, but they resemble ODP enterprise language and use XML-style notations (see [10] and [2]).

In the service publication process (step 1), service providers send service offers to the service offer repository, to state claims about the type and properties of the

services. A service offer describes functional and non-functional properties of the service to be published: the actual service interface signature, service behaviour, requirements for technical bindings (e.g. transport protocol), and attributes such as service quality and trust related commitments. The service offer repository then initiates a conformance validation process. For this purpose, a service type corresponding the claimed service type is retrieved from the type repository (step 2). The service type defines syntactical structures for service interface signature and messages, externally visible service behaviour and semantics for exchanged messages [2]. Conformance validation is executed by the service type repository holding the corresponding service type (step 3). Only after a successful validation, the service offer is published (step 4a), otherwise a service typing mismatch is reported between the service offer and its claimed type (step 4b).

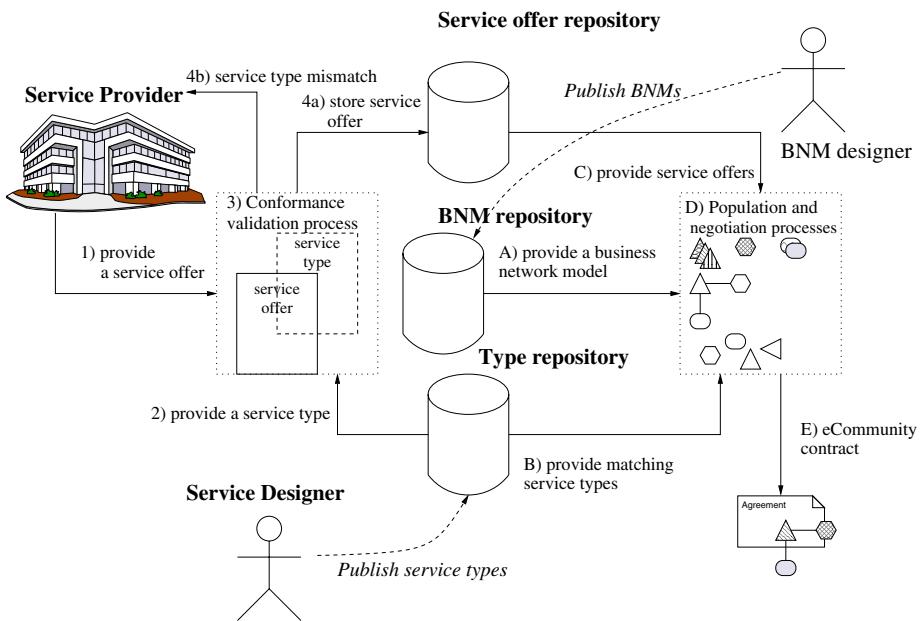


Fig. 3. Repository usage during eCommunity life-cycle

When an eCommunity establishment process is initiated by a willing partner, the corresponding business network model is first fetched from the BNM repository (step A). The population process (step D) provides a set of interoperable eCommunity proposals where roles of the BNM are filled with potential partners. For this purpose, the type repository is consulted for providing service types matching the requirements of the business network model (step B), after which the service offer repository can be used to provide the corresponding service providers (step C). After population, and the subsequent negotiation, the eCommunity contract is received (step E) and distributed to every participant.

The service interoperability and correct operation of the community assumes that the metalevel information on BNMs, service types, and service offers is correct. Therefore, we find it necessary to collect the metainformation into repositories, where the trustworthiness of the information source can be controlled, and the quality of the information can be validated by the repository management actions. These aspects must be weaved into the tasks involved with eCommunity establishment, such as service publication or discovery [11, 5].

4 Operational-Time Environment

The operational-time environment comprises of the business process management agents maintaining the metalevel agreement of the collaboration, the local service management facilities at each enterprise, the monitoring embedded to each communication channel in each business network, and the business models used for resolving collaboration-level exception states. Section 5 follows with implementation detail.

4.1 eCommunity Management by Collaborative Agents

The community life-cycle includes steps for establishment (population and negotiation at the conceptual level, establishment of the community at the technical level), termination, reacting to change requests, and resolution of breaches. The eCommunity life-cycle is presented in Fig. 4.

The state transitions are performed by middleware agents. The eCommunity is placed into the initial state (populated) by the populator agent in the breeding environment. For other state transitions the responsibility is on the agents in the operational environment, Business Network Management Agents (BNMA, agent) and the eCommunity contract object, in collaboration. At each administrative domain, there is a BNMA agent responsible for managing the inter-organizational coordination and management protocols, global state information management, community participant management and contract breach management. The contract object is responsible for making decisions for the community it represents (currently, most decisions are referred to humans).

In the populated state, the BNMAs see a set of potential eCommunity contracts where the partners have a matching view of the business network model, a non-empty set of options for policy values representing communication and information representation aspects of the collaboration, and a matching set of requirements for platform services. The first contract draft is available to the initiating partner only, while in the in-negotiation state the suggested eContract is under the consideration of all participants. During the negotiations, the eContract can gain further decisions on joint policies and technologies in use, as described below. The technical establishment phase involves the local service management facilities. When unwanted situations are detected by the monitoring system and BNMAs agree that the case is a major fault, a reorganization process is started, potentially causing changes in the partnership. In addition, the

business network model involved may include epochs; an epoch change captures a major reorganization of the collaboration structure.

The negotiations are implemented as a coordinator-driven n-to-n negotiation. The coordinator is elected during the first negotiation round among the participant candidates. At each negotiation round the whole contract is sent to all candidates for consideration of the terms of the contract, with responses of agreement, disagreement, and possibly counter-offers. The coordinator merges the requested changes and proceeds to another negotiation round until all participant candidates agree on the contract terms or terminates the negotiation in lack of agreement. When any of the candidates refuse to participate the negotiations, the suggested eCommunity is moved back to population state.

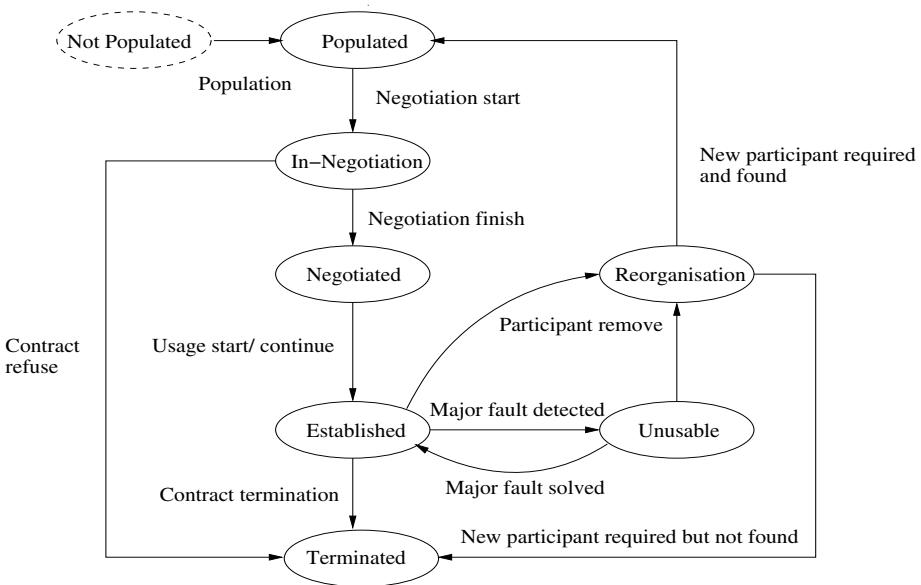


Fig. 4. Life cycle model of the community

The negotiations involve two categories of decisions. First, the business network models can describe alternatives for joint behaviour, and a policy decision needs to be made which alternative is used (for example, whether services must be prepaid or not). Second, some technical details such as information representation formats need to be agreed on. However, all business rules directing the local behaviour at each enterprise are not negotiated, only those defined in the business network model. To illustrate a common negotiation process, we assume there are three possible technology solutions for communication channels in a proposed eCommunity. The partner in the coordinator role sends the three-way proposal of the eCommunity contract to other participants. The participants responds with counter-offers containing the acceptable choices of technology to

them. As the counter-offers give different sets, it is the coordinator's task to find a cut of the sets and make the final decision within that set.

After a successful negotiation cycle, the contract is established by sending an establish request to all participants of the eCommunity. At this signal the provided services are prepared and monitors are configured with contextual metadata derived from the contract. The participants respond to the coordinator when their eCommunity elements are set up.

The eCommunity contract is a distributed object. Only the metainformation contents of the contract is distributed (because of heterogeneous technologies), and the distribution is embedded to the negotiation protocol of eCommunity contract. The local contract copies are kept in loose synchrony through BNAs.

The messages for state management and breach notifications form a basis for a reflective mechanism that keeps the meta-level information in the eCommunity contract and the actual service provision at the involved sites in synchrony. These messages cause changes in the contract contents, assuming that the change request are not contradictory. Changes in meta-data trigger activities for checking whether community participants need to be notified or requested a local act.

Each site has a local service management agent that holds and uses knowledge about locally deployed services and their various management methods [1]. The local management interfaces are homogenized by a protocol for requesting the system to prepare for running a service (resourcing), querying about communication points, and releasing the service. For local service-management services we propose to use generic service factories so that the actual service platform is irrelevant to the agent. Service factory can then be called with simple operations like *startService* to start a service or *stopService* to stop a service when it is not needed. Error reports allow the local management services to determine local and remote failures and to adhere to the agreed behaviour. This information can be used to improve performance of the observing domain and to file error reports to other domains in the community.

The local service management interface allows BNAs to collaboratively manage the community consistency. For example, initiative to replace or move a participant of the eCommunity causes requests to change the service point to the new participant's location, and recreate the bindings between new locations.

Besides indirect management by BNAs, the local services are controlled by local enterprise policies (i.e., business rules). Each enterprise is expected to have a private policy repository that captures rules for accessing services and distributing documents. These resource guards can be implemented in a similar style as has been presented in other policy-based management work considering also deontic policies [12, 13]. Each resource (processing unit, document) is governed by a monitor that consults the local policy repository for permission to proceed with a requested interaction. The local policies may change during the operational time of an eCommunity, and the local policies may override all community commitments. This may lead to policy conflicts during the eCommunity operation. Although the conflict styles identified are similar to static analysis approaches (see for example [14]), we start dynamic conflict resolution

by situations triggered by the monitoring system. A prototype implementation addresses problems of mismatch between organizations on access permissions, prohibitions, and obligations [15].

Communities can be terminated in a natural or forced way. A natural termination takes place if the contract is expired or the specified amount of sessions in the contract is exceeded. Contract session is specified as a one execution of the community functionality as described in the contract. The contract termination is forced if it is a consequence of a resolution process.

At any time, a participant may request that another participant in the community is removed, or inform others that it withdraws itself. The community contract defines the compensation process for such an event. The request naturally causes a negotiation cycle amongst participants. After the participant has left the eCommunity, the remaining participants will hold an election, lead by the coordinator, to decide if they will find a new participant to fill the now vacant role or terminate the community.

4.2 Context-Aware Monitoring

Monitoring is performed locally by each participant of a community, at the communication channel end-points. The monitors continuously evaluate whether observed behaviour is conformant to the expected behaviour explicated in the eContract. For example internal policies of organizations, service evolution and technical failures are causes for dynamic errors only detectable by active runtime monitoring. The monitors report progress of local business processes and detected breaches to local BNMs; if needed, the BNMs negotiate about required corrective actions.

Monitors are configured by BNMs with context information and related rules retrieved from the eContract. The eContract carries information on current progress state of the collaborative business process, and requirements for the correct progress of collaboration (service choreography), as well as process models for exceptional situations. The context relevant for the monitor represents an active part of the progression: the expected choreography, and freely designable monitoring criteria.

A monitor follows the behaviour of a service against the service choreography (external business process) represented by a two level state-machine, where the upper level represents task groups, and the lower level represents interrelated messages within that group. The upper-level machine is used to provide a coarser view to the progress through the choreography [16]. The task model is quite close to the work unit model in WS-CDL [17]; however, the model is not used for execution but for observing conformance. The state machine notifies completion of a task once all expected messages in the task are exchanged in an acceptable order. Problem notifications can be raised for order breaches, missing messages, and information contents. The monitoring facilities use the lower level only internally and reports to local BNM using the task level. Reports from the monitors include meta-events when a specific task is completed and when behaviour of the services are not correct.

The steps taken when a message is sent between two service applications are shown in Fig. 5. The illustration also indicates the intercepting location of the monitors in the communication channel architecture. In the figure, the box WS-Tr. represents additional services in the channel, such as aspects of distribution transparency and transaction support (for more details, see [5]).

1. Application sends a message
2. Monitor intercepts the message
3. Monitor checks compliance to context and rules
- 4.a IF message was ok: let message go out of org.
- 4.b ELSE report fault to BNMA
5. Monitor intercepts the message
6. Monitor checks compliance to context and rules
- 7.a IF message was ok: let message go to application
- 7.b ELSE report fault to BNMA
8. Application receives message

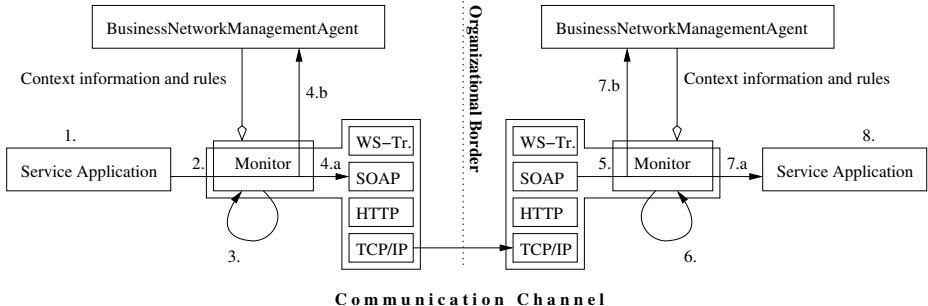


Fig. 5. Monitor as a part of the communication channel during a message send

The task level state information is not aggressively distributed to all participants, but relayed to the contract object to be retrieved through it by those participants needing it, when needed. Still, the level of traffic generated may cause undue overhead unless business network structure is reasonably designed. The task boundaries are annotated on the choreographies by the designers, and the analysis of the models should therefore include also the cost of the model.

In the monitoring criteria, it is possible to use rules that consider the business network status as well, for catching behaviour rules such as "payment must be received by the bank before the warehouse can ship the delivery". The monitors can also control aspects of information representation, trustworthiness of the service requests, and other nonfunctional aspects of the collaboration.

4.3 Breach Management and Epochs

Breach management is triggered by the monitors by notifying their local BNAs both of minor and major discrepancies. The BNA decides whether the discrepancy is to be considered a breach, or can be passed with local recovery actions or by ignoring the occurrence. When a breach is detected, the detecting BNA notifies the eCommunity coordinator with information on the event and the participant considered responsible of the failure. For the resolution of a serious breach, the eCommunity enters an intermediate state during which decisions are taken (potentially negotiated) on the corrective actions.

Architecturally, the recovery scenarios should not be fixed into the agents, but need to be derived from the business network model repository and become part of the eContracts in the eCommunity establishment phase. The recovery processes are one of the elements to be either matched in population or negotiated thereafter. However, in the current prototype, the offender can either admit or deny the breach. For admitted breaches, the compensation processes agreed in the eCommunity contract are used, and the activities of the eCommunity can be continued normally. For denied breaches, human intervention is used, and potentially leads to change of the faulty participant. The agent provides a method *changeParticipant* that invokes a negotiation on whether the offender needs to be excluded from the eCommunity, and subsequently involves the populator to assist in selection and interoperability assurance for a new participant. Alternatively, the breach can lead to total termination of the eCommunity after negotiations.

The transition to the separate resolution process requires that the involved service providers are prepared to run additional, infrastructure-level processes in addition to the original business process. The enterprises are expected to provide business facilities able to respond to for example sanction negotiations, thus conforming to a best-practices expectation. In addition, at the middleware level, facilities for epoch management are required.

An epoch is defined as a period during which roles and services of the network participants are stable. Two subsequent epochs can have different sets of roles and services involved, and between epochs transition rules can be defined. Participants in certain roles are required to reappear in a specific role in the next epoch, while some others leave the community. Transition between two epochs require synchronization between partners.

5 Prototypes and Lessons Learned

The web-Pilarcos middleware prototype is implemented in Java, with a mix of J2EE technology and standard Java objects. The prototype is built using JBoss and services are distributed as Web Services. The organization-oriented middleware services (Contract object, Contract Repository, NetworkManagement-Agent, and Monitors) are implemented with J2EE; the public domain services (Type Repository, Populator, Service Offer Repository, and BPM Repository) are implemented as standard Java objects. All components, except the eContracts residing in the Contract repository, are accessible through Web Services interfaces, either within an enterprise, or across enterprise boundaries. The most important components of the implementation include the eContract, the BN-MAs, and the monitors. The set of prototype services also includes an application for visualizing the inter-enterprise business process and its progress [18]. This application provides human access to the partner change and eContract renegotiation methods through BNMA.

The NetworkManagementAgent component implements BNMA interfaces for eContract life-cycle management, negotiation, establishment, global state man-

```

Lifecycle management interface:
String[] populateArchitecture(String architectureName, String myRoleName,
                               int maxOffers, int maxTime,
                               String usedPolicies)
int[] negotiateContract(String contractID)
int[] instantiateContract(String contractID)
void terminateContract(String contractID)
String createNewSession(String contractID)

Negotiation and establishment interfaces:
acceptContract(ContractContent contract, String participant)
acceptContractResponse(String contract_id, String participant,
                       NegotiationResponse negotiation_response)
establishContract(String contractID, String participant)
establishContractResponse(String contractID, String participant,
                           boolean success)
renegotiateContract(String contractID)
renegotiateContractResponse(String contractID, String participant,
                            boolean renegotiate)
announceResult(String contractID, boolean renegotiate)
cancelContract(String contractID, String participant)

Global state management interface:
updateTaskState(String contractID, String sessionID, String taskID,
                 String newState, String participant)
epochChanged(String contractID, String sessionID, String newEpoch,
             String participant)

Monitor input interface:
updateEpochState(String sessionID, String roleID,
                  String epochID, String stateID)
sessionEpochFinished(String sessionID, String epochID)

Monitor configuration interface:
addSession(String sessionID, String[] epochIDs, String[] policyIDs,
           String[] myRoleIDs, String[] otherRoleIDs)
addRole(String sessionID, String roleID, String[] roleEpochIDs,
        String[] policyIDs)
addEpochAutomata(String sessionID, String roleID, String epochID,
                  StateAutomata epochAutomata)
addChoreographyAutomata(String sessionID, String roleID, String epochID,
                        StateAutomata choreographyAutomata)

setActiveSessionEpoch(String sessionID, String epochID)
deactivateSession(String sessionID)
isSessionActive(String sessionID)

deactivateRole(String sessionID, String roleID)
isRoleActive(String sessionID, String roleID)

updateEpochState(String sessionID, String roleID,
                 String epochID, String stateID)

```

Fig. 6. Interfaces of middleware services

agement, and monitor input. The first three interfaces are used between enterprises, the rest by local services. The BNMA interfaces are described in Fig. 6.

The BNMAs form an agent-style discussion amongst themselves: the initiator suggests a collaboration using a named model for a group of named partners, and the group members make counter-offers to the suggested details. The propositions are taken as believable facts (we have trust-management extensions planned, which change this). As an extension to the traditional agent discussions, the BNMAs are able to detect breaches to the agreed behaviour, and start negotiations on the caused situation. The BNMAs are not self-contained as agents,

as initiatives to actions are received by users, applications, and changes in the local computing services.

The monitors are hooked into the communication channel architecture, as proxies into the JBoss environment to intercept messages. In addition, monitors implement a service interface for metadata configuration. The monitor interfaces are described in Fig. 6.

The scalability of the architecture has been carefully analyzed, mainly resulting to aspects that need to be verified in the business network models used. As a consequence, the analysis gives us guidelines for providing new software engineering tools in the area of model verification and model property analysis.

Across enterprises, communication is restricted to global business network state updates, error resolving, and contract life cycle management. State updates are done only when epochs are changed and tasks completed, so the communication volume depends on the business network design. In the overall service and model production methodology behind our work, it is essential that the business network models are carefully verified and analyzed before publication. One of the essential features to analyze is the cost of the operation of the model. We expect that the task/epoch ratio is kept relatively low.

Within enterprises, the cost of communication is somewhat lower, especially if critical components are appropriately deployed. Monitoring cost is a scalability challenge, but can be partially overcome by suitable selection of monitoring modes: only proactive monitoring that prevents further steps in the business process interferes severely with the overall performance, and should be restricted to carefully selected features. Loose feedback loops from monitors to BNAs can also be used and still acquire an operational-time detection of frauds and failures in collaborations.

As the population process is essential for the feasibility of the presented architecture, the first phase prototype included only the population process [19, 20], and performance evaluation on that. Having restricted the complex constraint satisfaction problem appropriately, we found the performance mainly dependent on the number of roles in the network and policies per role [21].

6 Conclusion

The B2B middleware developed in web-Pilarcos provides support for autonomously administered peer services that collaborate in a loosely coupled eCommunity. The eCommunity management by design excludes the need for distributed enactment services, but in contrast provides facilities for ensuring interoperability at semantic and pragmatic level. In this respect the federated approach has a different focus from those in most other P2P community management systems, such as ADEPT [22] or METEOR [23], and contract-driven integration approaches, such as ebXML [9]. Even most virtual enterprise support environments, such as CrossFlow [24] and WISE (workflow-based internet services) [25], rely on models for distributed business process enactment. However, the web-Pilarcos approach leaves enactment as a local business processing task, concentrating on interoperability monitoring.

The web-Pilarcos concept of eContracts ties together ICT related viewpoints of ODP (Open Distributed Processing reference model [26]), also ranging to some features of business aspects. The ODP-RM introduces information, computational, engineering and technical viewpoints. Each of these present interrelated but somewhat independent aspects of the collaboration features and its composition using more basic computing services. The web-Pilarcos contract structure captures these aspects in its BNMs, binding requirements, and behavioural and non-functional monitoring rules [10]. In other projects, like BCA [27], contracts have legal and business level focus and detect contract breaches post-operatively [28]. The web-Pilarcos aims for more real-time intervention.

In the web-Pilarcos middleware, the eCommunity life cycle is built to be collaboration-process-aware. The architecture model acts on two abstraction layers, the upper layer involved with abstract, external business process describing the collaboration requirements; the lower layer comprised of actual services bound to the eCommunity dynamically. In this kind of environment, static verification of models and interoperability cannot be complete. In the B2B middleware provided by the web-Pilarcos project, we find it necessary to develop control environments for monitoring and reflectively restructuring the operational eCommunities, besides a breeding environment. The goals are similar to other projects, but the solution methods differ. While ADEPT supports direct modification of the workflow control structures, web-Pilarcos uses negotiated policy-values to choose between predefined behaviour alternatives. The web-Pilarcos solution even requires that well-formed contracts include suitable recovery processes that involve whole communities. In contrast to METEOR-S, the web-Pilarcos platform has no central tool for making the whole of interoperability analysis, but partial static verification is done at the meta-data repositories, and monitoring is used to detect further problems.

The B2B middleware is in some extent comparable to agent-based approaches, such as MASSYVE [29]. The main difference seems to be the separation of business-application services and B2B middleware services from each other. The web-Pilarcos middleware agents do not provide workflow execution, but expect local application management to play that part. In contrast to [30], the middleware agents are responsible of semantic verification and failure resolution, and use separate monitors to help and report.

The web-Pilarcos middleware increases the ability of an enterprise to adapt to changes at strategical business processes, platform technologies, and partners and partners' services within the business networks. The presented middleware services indicate the essential B2B services on which to invest, in order to decrease the cost and reimplementation effort caused by changes in the operational environment. The operational environment of web-Pilarcos described in this paper enhances our earlier work on collaboration partner matching in the Pilarcos project [19] by introducing the monitoring of business processes and local enterprise policies, and by providing a set of eCommunity management protocols at the meta-information level.

The provision of the web-Pilarcos architecture requires further development of business process modeling techniques. The collaboration of business processes or workflows should be modeled without unnecessary revealing of local processing steps. Instead, only the collaborative part (external view) should be agreed on and monitored. Work is already started by the component-driven approach on splitting workflows into Web Services. The structural needs of business process models are also widened by the requirements of incorporating reusable sanctioning, recovery, and compensation processes into eCommunity contracts. Furthermore, shared ontologies and repositories for business process models should be made available. Such facilities would improve the potential for reaching interoperability in an environment where service components are truly developed independently from each other. More fundamentally, ontologies and repositories would create a facility for checking semantical similarity of business process model as part of the interoperability tests during eCommunity establishment.

Acknowledgment

This article is based on work performed in the Pilarcos and web-Pilarcos projects at the Department of Computer Science at the University of Helsinki. The Pilarcos project was funded by the National Technology Agency TEKES in Finland, Nokia, SysOpen and Tellabs. In web-Pilarcos, active partners have been VTT, Elisa and SysOpen. The work much integrates with RM-ODP standards work, and recently has found an interesting context in the INTEROP NoE collaboration.

References

1. Kutvonen, L.: Automated management of inter-organisational applications. In: Proc. 6th International Enterprise Distributed Object Computing Conference (EDOC2002). (2002)
2. Kutvonen, L., Ruokolainen, T., Metso, J., Haataja, J.: Interoperability middleware for federated enterprise applications in web-Pilarcos. In: Int. Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'05), Springer Verlag (2005)
3. Kutvonen, L.: Relating MDA and inter-enterprise collaboration management. In Akehurst, D., ed.: Second European Workshop on Model Driven Architecture (MDA), University of Kent (2004) 84–88
4. Ruokolainen, T.: Component interoperability. Master's thesis, Department of Computer Science, University of Helsinki (2004) In Finnish.
5. Kutvonen, L.: Trading services in open distributed environments. PhD thesis, Department of Computer Science, University of Helsinki (1998)
6. OASIS Consortium: Universal Description, Discovery and Integration of Web Services (UDDI) 3. (2002) http://uddi.org/pubs/uddi_v3.htm.
7. ISO/IEC JTC1: Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. ODP Trading Function. (1997) IS13235.

8. ISO/IEC JTC1: Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. ODP Type Repository Function. (1999) IS14746.
9. Kotok, A., Webber, D.R.R.: ebXML: The New Global Standard for Doing Business Over the Internet. New Riders, Boston (2001)
10. Kutvonen, L.: Challenges for ODP-based infrastructure for managing dynamic B2B networks. In Vallecillo, A., Linington, P., Wood, B., eds.: Workshop on ODP for Enterprise Computing (WODPEC 2004). (2004) 57–64
11. Viljanen, L., Ruohomaa, S., Kutvonen, L.: The TuBE approach to trust management. In: Proceedings of the 3rd iTrust internal workshop. (2004)
12. Kollingbaum, M.J., Norman, T.J.: Supervised interaction: creating a web of trust for contracting agents in electronic environments. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, ACM Press (2002) 272–279
13. Lymberopoulos, L., Lupu, E., Sloman, M.: An adaptive policy based framework for network services management. *Journal of Network and systems Management* **11** (2003) 277–303 Special issue on Policy based management.
14. Dunlop, N., Indulska, J., Raymond, K.: Dynamic conflict detection in policy-based management systems. In: 6th International Enterprise Distributed Object Computing Conference (EDOC2002). (2002)
15. Karppinen, M.: Distributed policy enforcement. Master's thesis, Department of Computer Science, University of Helsinki (2003) In Finnish.
16. Haataja, J.: Monitoring of inter-enterprise collaboration networks in Web-Services environments. Master's thesis, Department of Computer Science, University of Helsinki (2005) In Finnish.
17. W3C: Web Services Choreography Description language. (2004) <http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/>, Working draft.
18. Henriksson, R., Kare, A., Lähde, M., Mäki, A.J., Stenberg, M., Virtanen, T.: Business network management GUI. <http://www.cs.helsinki.fi/group/ohtu/s-2004/ltv.html> (2004) Software engineering project.
19. Vähääaho, M., Haataja, J.P., Metso, J., Suoranta, T., Silfver, E., Kutvonen, L.: Pilarcos prototype II. Technical Report C-2003-12, Department of Computer Science, University of Helsinki (2003)
20. Vähääaho, M.: Trading with architecture models. Master's thesis, University of Helsinki (2003) In Finnish.
21. Kutvonen, L., Metso, J.: Services, contracts, policies and eCommunities – Relationship to ODP framework. In: Workshop on ODP for Enterprise Computing (WODPEC 2005), IEEE Digital Library (2005)
22. Reichert, M., Dadam, P.: Adeptflex – supporting dynamic changes of workflow without losing control. *Journal of Intelligent Information Systems* **10** (1998) 93–129 Special Issue on Workflow Management.
23. Aggarwal, R., Verma, K., Miller, J., Milnor, W.: Constraint Driven Web Service Composition in METEOR-S. In: Proceedings of the IEEE SCC. (2004)
24. Grefen, P., Aberer, K., Hoffner, Y., Ludwig, H.: CrossFlow: Cross-Organizational Workflow Management in Dynamic Virtual Enterprises. *International Journal of Computer Systems Sciences and Engineering* **15** (2000) 277–290
25. Lazcano, A., Alonso, G., Schuldt, H., Schuler, C.: The WISE approach to Electronic Commerce. *Int. Journal of Computer Systems Science and Engineering* (2000)
26. ISO/IEC JTC1: Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. (1996) IS10746.

27. Milosevic, Z., Limington, P.F., S.Gibson, Kulkarni, S., J.Cole: Inter-organisational collaborations supported by e-contracts. In: The fourth IFIP conference on E-commerce, E-Business, E-Government, Toulouse, France (2004)
28. Quirchmayr, G., Milosevic, Z., Tagg, R., Cole, J., Kulkarni, S.: Establishment of virtual enterprise contracts. In: Database and Expert Systems Applications : 13th International Conference. Volume LNCS 2453., Springer-Verlag (2002) 236–
29. Rabelo, R., Camarinha-Matos, L.M., Vallejos, R.V.: Agent-based brokerage for virtual enterprise creation in the moulds industry. In: E-business and Virtual Enterprises. (2000) <http://gsigma-grucon.ufsc.br/massyve>.
30. Daskalopulu, A., Dimitrakos, T., Maibaum, T.: Evidence-based electronic contract performance monitoring. The INFORMS Journal of Group Decision and Negotiation (2002) Special Issue on Formal Modelling in E-Commerce.

DOA 2005 PC Co-chairs' Message

Welcome to the Proceedings of the 2005 International Conference on Distributed Objects and Applications (DOA). Some of the world's most important and critical software systems are based on distribution technologies. For example, distributed objects run critical systems in industries such as telecommunication, manufacturing, finance, insurance, and government. When a phone call is made or a financial transaction performed, chances are that distributed objects are acting in the background. Although existing distribution technologies, such as CORBA, DCOM and Java-based technologies have been widely successful, they are still evolving and serving as the basis for emerging technologies and standards, such as CORBA Components, J2EE, .NET, and Web Services. Regardless of the specifics of each approach, they all aim to provide openness, reliability, scalability, distribution transparency, security, ease of development, and support for heterogeneity between applications and platforms. Also, of utmost importance today is the ability to integrate distributed object systems with other technologies such as the web, multimedia systems, databases, message-oriented middleware, the Global Information Grid, and peer-to-peer systems. However, significant research and development continues to be required in all of these areas in order to continue to advance the state of the art and broaden the scope of the applicability of distribution technologies.

The paper selection process was highly competitive with an acceptance range of 29.5.

The final program spans the following DOA-related topics: Web services and service-oriented architectures, multicast and fault tolerance, aspect-oriented middleware, component middleware, Java environments, mobility support, techniques for application hosting, data persistency, security and privacy, messaging and publish/subscribe, and communication services. We would like to thank the authors of the submitted papers for their time and effort in making for a very competitive selection process, the program committee members for their diligence in reviewing the submissions, the General Chairs and other members of the organizing committee, and the attendees for making DOA 2005 a success.

August 2005

Ozalp Babaoglu, University of Bologna
Arno Jacobsen, University of Toronto
Joe Loyall, BBN Technologies
(DOA 2005 Program Committee Co-chairs)

Developing a Web Service for Distributed Persistent Objects in the Context of an XML Database Programming Language

Henrike Schuhart, Dominik Pietzsch, and Volker Linnemann

Institut für Informationssysteme,
Universität zu Lübeck,
Ratzeburger Allee 160, D-23538 Lübeck, Germany
`{schuhart, pietzsch, linnemann}@ifis.uni-luebeck.de`

Abstract. The development of data centric applications should be performed in a high-level and transparent way. In particular, aspects concerning the persistency and distribution of business objects should not influence or restrict the application design. Furthermore applications should be platform independent and should be able to exchange data independently of their programming language origin.

There are several approaches for an architecture for distributed objects. One example is CORBA. JDO and EJB allow specifications for distributed persistent objects offering transparent persistency up to a certain degree. Nevertheless, the programmer is still forced to write explicit code for making objects persistent or for connecting to distributed objects.

In contrast to existing approaches, the **XOBEDBPL** project develops a database programming language with transparency with respect to types, and persistency and distribution with respect to objects. Application development is performed on a high-level business object level only. A web service for realizing distributed persistency and data exchange is internal and completely integrated in the **XOBEDBPL** runtime environment. Although the **XOBEDBPL** language is an extension of the Java programming language, the introduced concepts could be easily transferred to other object-oriented programming languages.

1 Introduction

Today's programming languages and tools generally do not allow transparent development of data centric applications. This means that, though it should not, it indeed makes a difference for the programmer whether application objects are kept persistently or have transient lifetime and whether they are local or shared. Due to insufficient abstraction levels, the same solution is often redeveloped. It would be preferable to be able to develop applications on the business object level without having to think about persistency all over the program. Furthermore, business object models should not be restricted due to potential persistency or distribution. Instead, these aspects should be solved automatically, meaning that code for persistency and distribution should be system generated. Another important aspect is the exchange of objects between different applications implying a transparent, platform-, language- and type-independent process as well.

Besides CORBA [1], which does not support persistency explicitly, JDO [2] and EJB [3] provide persistent, slightly restricted objects for the Java programming language. In JDO and EJB the programmer has to connect to the persistency instance explicitly and has to take care by method invocation, for example, that certain objects become persistent. JDO persistent objects are not language independent, while in EJB every persistent object can be accessed via an integrated web service interface. For every type there is a different WSDL definition and an object interface for this web service must be generated each time. Hibernate [4] offers an O/R mapping framework for Java environments, meant to shield most common object-relational-data-persistency problems from the developer. Nevertheless, the programmer is forced to explicitly define how to load and store objects of a persistent class. This is done by the Hibernate mapping file telling Hibernate what table in the database it has to access, and what columns in that table it should use. In Java Spaces [5] objects must be made persistent explicitly. Java Spaces systems do not provide a nearly transparent persistent/transient layer, and work only on copies of entries.

The **XML OBjEcts DataBase Programming Language (XOBE_{DBPL})** is designed as a high-level programming language offering static type checking, transparent object states and distribution. The **XOBE_{DBPL}** programmer writes applications on a business-object-level only. The predecessor project **XOBE** introduces statically type checked XML objects, XPath queries and update expressions. A first version of **XOBE_{DBPL}** overcomes transient only states for objects in **XOBE** and integrates type independent and transparent persistency.

Previous Work. In [6] and [7] we present an extension of Java by XML objects, XML language descriptions and XPath expressions for read-only queries. Moreover, the static type checking mechanism with an efficient algorithm is introduced. Recent papers of the authors about **XOBE_{DBPL}** [8],[9] overcome, among others, readonly queries for XML objects by introducing update expressions. Updates are also statically type checked. In [10] the transparent type independent persistency concept for objects is introduced. The persistency concept is transparent in the sense that a programmer does not have to know whether the objects he is working with are persistent or transient. Moreover, the persistency concept is type independent in the sense that any object regardless of type can become persistent.

Contribution of this paper. The main contribution of this paper is the realization of a transparent distribution concept of persistent objects in **XOBE_{DBPL}**. The concept is transparent in the sense that implementational or technical details are hidden from the programmer. In **XOBE_{DBPL}** we use a web service which is completely encapsulated in the **XOBE_{DBPL}** runtime environment and can be seen as a server for distributed and persistent data. **XOBE_{DBPL}** web service instances may exchange data and information. Besides presenting the concept, we present the web service and implementation of the **XOBE_{DBPL}** runtime system in detail. An application example shows the feasibility of our approach.

The paper is organized as follows. Section 2 gives an overview over the **XOBE_{DBPL}** project. Section 3 introduces the architecture used in **XOBE_{DBPL}**. In section 4 we present our new realization of the distributed persistency concept by the **XOBE_{DBPL}**.

web service. Section 5 gives an application example showing a distributed paper archive for a research environment. Finally in section 6 we present related work and finish with conclusions in section 7.

2 XOBEDBPL

In this section we review the main aspects concerning **XOBEDBPL** in an informal manner.

XML Integration. **XOBEDBPL** extends the object oriented programming language Java by language constructs for processing XML fragments and in particular XML documents. XPath [11] is used for traversing XML objects and update expressions allow to manipulate existing XML objects. In **XOBEDBPL** we represent XML fragments, e.g. trees corresponding to a given schema, by XML objects. Therefore XML objects are first-class data values that may be used like any other data value in Java. The declared schema, which can be either a DTD [12] or an XML Schema [13], is used to type different XML objects.

Throughout this paper we use an application example modeling a bibliography. A bibliography manages a collection of articles, inproceedings, books etc.. Listing 1 shows the corresponding **XOBEDBPL** class definition. To demonstrate how XML is integrated in **XOBEDBPL**, we declare the member variable `bib` in line 4 to be of the XML type `dblp`. This type is defined in the imported DTD called `dblp.dtd` in line 1. The `ximport` statement in **XOBEDBPL** works analogously to Java's import declaration. The DTD is related to the DBLP project [14]. The DBLP server provides bibliographic information on major computer science journals and proceedings. DBLP stands for *Digital Bibliography & Library Project*.

```

1  ximport dblp.dtd;
2
3  public class Bibliography{
4      dblp bib;
5      String description;
6
7      public void addWWW(String key, String date, String title, String url){
8          xml<www> internet = <www key={key} date={date}>
9              <title>{title}</title>
10             <url>{url}</url>
11         </www>;
12         add(internet);
13     }
14
15     public void add(xml<((article | inproceedings | proceedings |
16                           book | incollection | phdthesis |
17                           masterthesis | www)> publication){
18         $UPDATE bib INSERT publication$;
19     }
20
21     public xml<(article)*> searchArticles(String author){
22         xml<(article)*> articles = $bib/articles[author={author}]$;
23         return articles;
24     }
25 }
```

Listing 1. Class version of the bibliography

The method `addWWW` in lines 7-13 uses an XML object constructor to construct an XML object of type `www` with the given parameters, e.g. title, url. Lines 15-19 exemplify **XOBE_{DBPL}**'s update expressions by showing an insert operation. In addition to inserts **XOBE_{DBPL}** offers deletions, replacements, renaming and combinations of them. The update in line 18 inserts a new publication into the content of the bibliography member variable `bib`. Finally queries can be performed upon XML objects by XPath expressions. In line 22 all articles published by a given author are searched and returned. An XPath result in **XOBE_{DBPL}** is always a list of XML objects. If XML types are unambiguous, neither a list nor a choice of several types, the keyword `xml` may be omitted, as it is done in line 4, otherwise the keyword `xml` is followed by brackets containing a list or choice of XML types. Static type checking for XML objects and operations in **XOBE_{DBPL}** consists of three main parts, namely XML formalization, type inference and subtype checking. A more detailed description of the syntax and semantics of XML objects in **XOBE_{DBPL}** and of static type checking can be found in [6] and respectively in [8].

Persistency. Up to now XML objects as well as Java objects are transient meaning that these objects have only application lifetime, i.e. data gets lost each time an application finishes. In contrast to tools offering type dependent persistency where objects of some types can become persistent while others not, **XOBE_{DBPL}** offers transparent and type independent persistency by introducing a persistent environment called `database`. In case of persistency frameworks class declarations often have to fulfill certain conditions, e.g. descriptor files are needed listing persistent capable types as it is done in the object oriented database system Fast Objects [15].

```

1  ximport dblp.dtd;
2
3  public database Bibliography{
4      dblp bib;
5      String description;
6
7      // everything else remains unchanged
8      // ...
9  }
```

Listing 2. Persistent version of the bibliography

```

1  public class MainSearch{
2      public static void main(String [] args){
3          ...
4          //Search for the specific bibliography(s)
5          String description = "Computer Science Bibliography";
6          List bibliographys = $Bibliography[description={description}]$;
7          Bibliography theBibliography = null;
8          //if there is at least one, get the first
9          if(bibliographys.size()>0)
10         theBibliography = (Bibliography) bibliographys.get(0);
11         //do anything with the bibliography
12         ...
13     }
```

Listing 3. Searching and accessing an already existing (persistent) bibliography object.

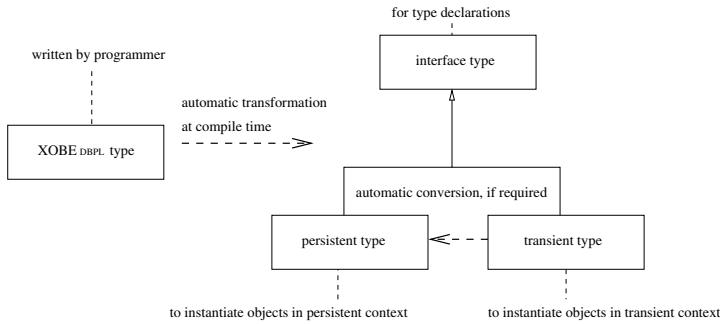


Fig. 1. The concept to realize type independent persistency in **XOBEDBPL**

A database declaration in **XOBEDBPL** is used analogously to class declarations well known from Java. The most important difference is that the keyword **database** implies that generated objects of this database implicitly become persistent.

Member variables of such database declarations become persistent by reachability **regardless of type** and without modifying former defined class declarations. By defining the bibliography objects within a persistent environment, its objects will be kept persistent. From the programmer's point of view it is sufficient to declare the bibliography declaration in listing 1 as **database**, everything else is done automatically. Constructors generate new persistent objects and method invocations upon these objects persistently modify them. In listing 2 the persistent version of the bibliography is shown. To retrieve an already existing persistent object **XOBEDBPL** provides an XPath expression searching for all objects of a given class with given member variable values. An example for retrieving a bibliography object with description *Computer Science Bibliography* is given in listing 3 in lines 5 and 6.

The basic concept to realize the type independent and transparent persistency concept in **XOBEDBPL** is that every class declaration is automatically transformed into a transient and a persistent type variant, both implementing a common interface.

As illustrated in figure 1, the interface type is used throughout the transformed code, while the transient type is instantiated in transient environments and the persistent type is instantiated in persistent environments. In case an instantiated transient object is assigned later on to a variable within a persistent environment, it is transformed into an equivalent persistent object. Detection and transformation into a persistent object are performed automatically. The generation process of interface, transient and persistent types is done at compile time. Some implementation details are given in section 3.

Transactions. To guarantee consistent access to persistent objects, a transaction concept is needed. Basically any method invocation upon a persistent object implicitly modifies its state in a single transaction. If several method invocations should be summarized to one transaction, a **XOBEDBPL** transaction statement can be used. It is defined analogously to Java's already existing **synchronized** statement. The transaction statement requires a list of variable names. These variables must reference persistent objects. The transaction statement block is than executed as one transaction for all persistent objects being listed as parameters. At compile time these transactions are

automatically transformed into code which, among others, communicates with a local **XOBE_{DBPL}** server. This server is responsible for locking. Deadlocks are excluded, since affected persistent objects are predetermined by the required parameter list. Listing 4 gives an impression of complex transactions.

```

1   ...
2   transaction(bibliography ){
3     if(bibliography . searchArticles(author) . size ()>0{
4       //only if articles of the given author exist
5       //apply modifying operation on the bibliography
6     }
7 }
```

Listing 4. A complex transaction upon a (persistent) bibliography object

Up to now persistent objects are stored automatically by the generated code into a local database and loaded equivalently. The distribution aspect enabling to share data among applications on different clients is discussed in the next sections. While section 3 describes general realization concepts, section 4 explains details about the **XOBE_{DBPL}** web service.

3 Architecture

In this section we give some details about the architecture and implementation of the **XOBE_{DBPL}** web service in general and about **XOBE_{DBPL}** clients in particular. Figure 2 presents the structure of the **XOBE_{DBPL}** web service and clients. A client connects to a single **XOBE_{DBPL}** web service called **XOBE_{DBPL} Service Node** (XSN), which is part of a network consisting of one or more **XOBE_{DBPL} Web Services** (XWS) and one or more **BackGround Persistence Services** (BGPS). Such a network is called shard. A BGPS can be used directly by one ore more **XOBE_{DBPL}** web services of the same shard for storing data. Several shards build up a kind of local persistency grid. Consequently communication and organization aspects can be taken from grids as described in [16] and [17]. A **XOBE_{DBPL}** client is connected via a **XOBE** local server and described in the following paragraph. Besides **XOBE_{DBPL}** clients there can be any other web service client using the web service as a transparent, type independent, high-level distributed persistency service.

The **XOBE_{DBPL}** web service communication is completely integrated into the **XOBE_{DBPL}** runtime environment, which is hidden from the programmer as a whole.

Important aspects of a **XOBE_{DBPL}** client are shown in figure 3. The left hand side consists of the preprocessor which translates **XOBE_{DBPL}** programs into pure Java code. The resulting code includes, among other things, communication with the **XOBE** local server and therefore with the web service. The program parser is built with the help of the Java Compiler Compiler (JavaCC) [18]. The Xerces parser [19] is used for parsing DTDs and XML Schemas. Internally programs are represented using the Java Tree Builder (JTB) [20]. Within the preprocessor and respectively at compile time XML type checking occurs. The transformation process consists of two steps, first transformation of XML specific syntax parts and second transformation of type structure in general.

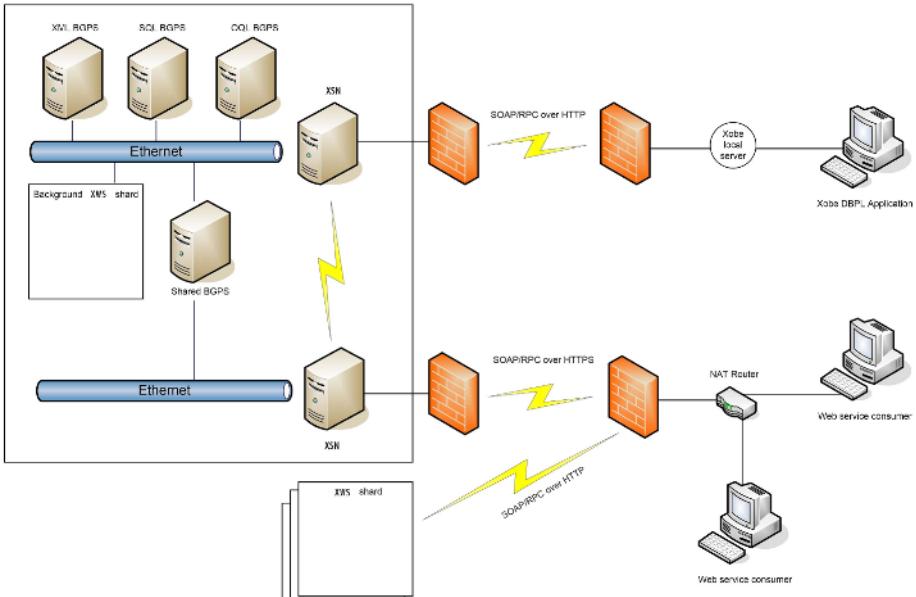


Fig. 2. Structure of the **XOBE_{DBPL}** web service environment

Transformation of type structure realizes transparent and type independent persistency and is done according to the concepts shown in figure 1. After the whole transformation process **XOBE_{DBPL}** programs can be executed by any Java Virtual Machine (JVM).

The left hand side of figure 3 illustrates a **XOBE_{DBPL}** client at runtime. Transformed **XOBE_{DBPL}** programs communicate with **XOBE** local servers. For local data they use local persistency mechanisms like databases. For shared data they can connect to one instance of the **XOBE_{DBPL}** web service. Communication with the web service is automatically achieved with the help of the WSDL2Java tool from the Apache Axis project [21].

At the moment we use Postgres [22] as relational database for performance reasons, but this is transparent to the programmer. Object-relational mapping is done automatically and independently of type. Principles from [23] are realized for this task. In the future we will test and use different databases including native XML systems and relational databases. Approaches which combine native XML and relational databases like the newly proposed hybrid system in [24] are very promising.

4 Web Service for Distributed Persistent Objects

In this section we introduce the **XOBE_{DBPL}** web service which realizes distributed persistent objects for the **XOBE_{DBPL}** database programming language. Entries for the DBLP project are desired to be read and inserted by different users working on different clients. More precisely, DBLP entries are supposed to be shared. In **XOBE_{DBPL}** writing a DBLP program is independent of this aspect, e.g. the program in listing 2 can remain

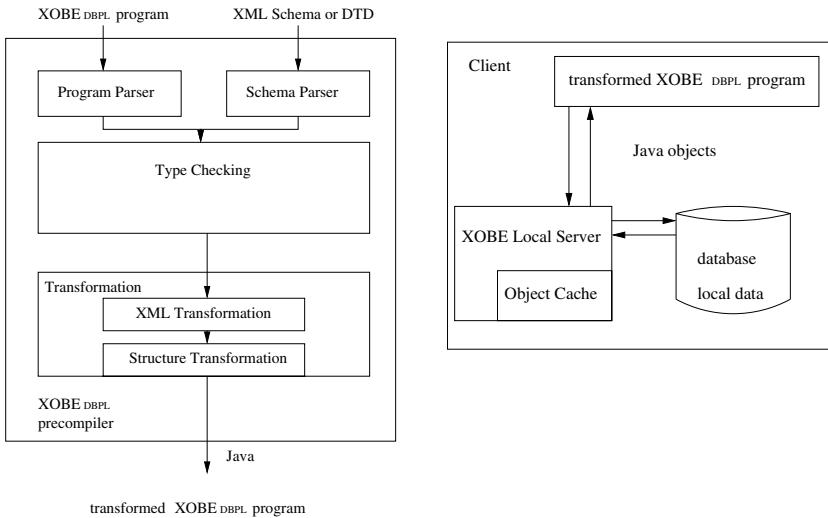


Fig. 3. Important constituents of a **XOBEDBPL** client. The left hand side shows the precompiler, the right hand side illustrates the runtime architecture of the client.

unchanged independently of entries being shared or being local and private. Nevertheless, somehow the runtime environment needs to know, if and where data should be stored. So, if the program of listing 2 is compiled without any further information, the runtime environment and the generated code will assume by default that all data is kept locally. Otherwise the programmer or an administrator has to write a small XML configuration file containing boolean information about distribution and one **XOBEDBPL** web service address. The referenced **XOBEDBPL** web service instance may be e.g. part of a local network in a business or part of a trusted network. Any persistent data of this client's runtime environment is interchanged directly with this web service instance.

The **XOBEDBPL** web service interface offers methods for all essential tasks including retrieving persistent objects, storing or respectively updating an object and deletions. Moreover, there are a few more methods solving tasks arising in the context of the former operations. The interface is kept thin and generic. Since the web service is completely integrated into the **XOBEDBPL** runtime environment and requests are formulated as well as responses are processed automatically, the interface must provide methods being capable to deal with objects of any possible type. Furthermore object references are represented with the help of ids. The web service's clients in the **XOBEDBPL** runtime environment are so called **XOBELocal servers** encapsulating communication with running **XOBEDBPL** programs. The latter are written by the programmer, although communication code is generated and added at compile time. The web service is completely hidden from the programmer. To understand this section, it is important to mention some further aspects concerning the **XOBEDBPL** runtime architecture. At first we use a limited number of web service instances, which are organized in a so called shard. Such a shard can be seen as a kind of local grid, where all web service hosts are con-

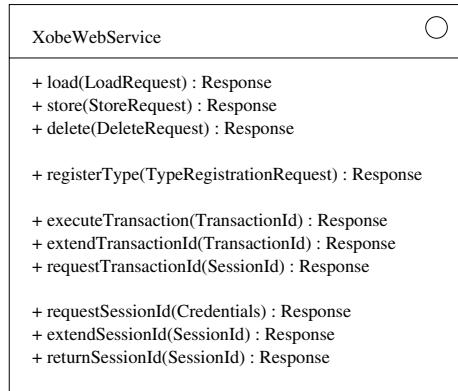


Fig. 4. Interface of the **XOBE_{DBPL}** web service

nected with each other, like in a business company. Every **XOBE_{DBPL}** web service client is directly connected with one specific host, usually the closest one. Objects are kept without redundancy. Besides the actual object data, the web service keeps meta information about its stored objects and types. Further details concerning the architecture can be found in section 3.

Figure 4 lists the available web service methods, which are predominantly `load` to retrieve persistent objects, `store` to store or respectively update a persistent object and `delete` to delete a persistent object. If a store request for an object of an unknown type occurs, the web service needs to know its structure. With the help of the method `registerType` the client can register this new type. A second important group of methods is provided for transactional support, e.g. `requestTransactionId`, `extendTransactionId` and `executeTransactionId`. Finally the last group consists of the methods `requestSessionId`, `extendSessionId` and `returnSessionId` providing the functionality to register (**XOBE_{DBPL}**) clients to the web service.

The **XOBE_{DBPL}** web service deals with five most important complex types, namely `DataObject` types to represent any object, `TypeDescriptor` to represent any object type, `id` types to represent object references and a hierarchy of result and response types. A `DataObject` can either be of an atomic kind, e.g. `String` and `integer`, or of a complex type. A `TypedObject` contains information about its type in the form of an `id` referencing a `TypeDescriptor`. `DataSets` are used to pass lists or arrays of objects. Finally an `AggregatedObject` contains additional information about its object `id`. Since objects can be exchanged without passing the whole referenced object tree, `DataReferences` are provided. `ComplexObjects` add object information about attributes. Any **XOBE_{DBPL}** object is transformed into a `DataObject` automatically. The upper part of figure 5 gives an overview of the data object hierarchy. A `TypeDescriptor` represents the type description of an object and consists of the corresponding type name and an `id` for this type descriptor. Furthermore a `ComplexTypeDescriptor` offers a list of the corresponding member variable declarations. If a complex type is derived from a super type, the corresponding

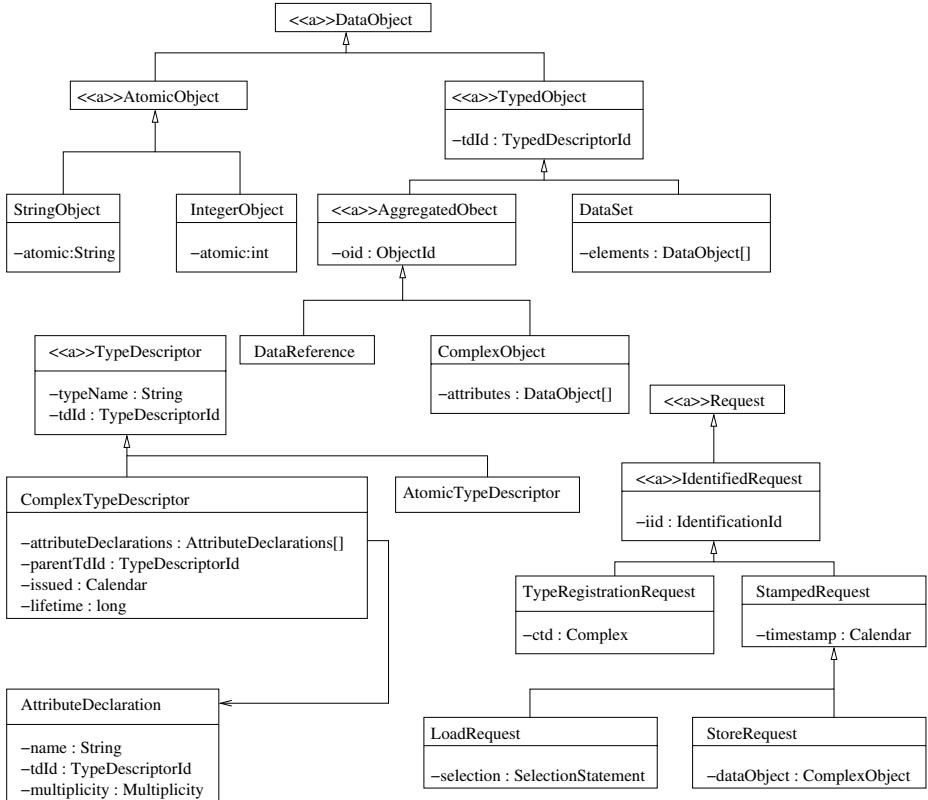
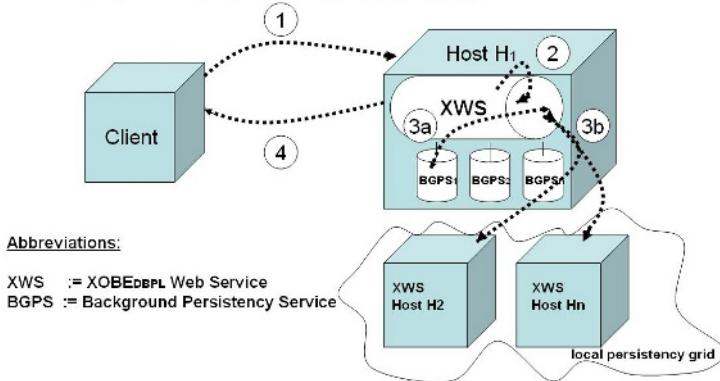


Fig. 5. Complex types for object data, type description and requests used by the **XOBEDBPL** web service

reference is given in form of a type descriptor id as well. In **XOBEDBPL** such type descriptions are generated for every type at compile time. The lower left part of figure 5 presents the type description hierarchy. Communication between **XOBEDBPL** clients and the web service enables to define a suitable request and response structure. The lower right part of figure 5 gives an overview of some important request types. According to the different operations offered by the web service, we have got a `LoadRequest` containing a selection statement used as parameter type for the load method. Similarly a `StoreRequest` contains an attribute holding the data object, which is going to be stored. `StoreRequest` is used as parameter type by the store operation. Other request types are defined accordingly. The response type hierarchy is designed according to the kind of response, e.g. a positive result, an error or a negative result. E.g. an expected answer in the context of a store request is the information that the objects have been stored successfully. In this case the response is of the concrete type `StoreDoneResponse`. Analogously the expected answer in the context of a load request contains the selected objects, which are then contained in the `LoadDoneResponse`. Other response types are used correspondingly. Communication with the web service starts with the client's registration. Received session ids enable to read object data. Moreover the session id

Actions:

1. send load request
2. check client's identification id and search metadata for selected object ids
- 3a. retrieve object data from local host
- 3b. broadcast request to registered XWS
4. receive and send response containing all found data objects

**Fig. 6.** Loading objects

allows to manipulate object data. Corresponding operations are executed as a sequence of independent transactions automatically. In case that a specific sequence of operations should be performed as one single transaction, the client is supposed to send a transaction id along with each single update instead. Transaction blocks in **XOBEDBPL** are converted at compile time to this kind of communication. Transactions are discussed later.

Loading. The **XOBEDBPL** web service receives a load request for either a single object or a set of objects. A load request consists of the client's session id and a selection expression. The web service checks the validity of the session id and continues to evaluate the selection expression. Selection expressions consist of object or type ids. With the help of the corresponding metadata, the web service tests, if the requested objects are available locally, otherwise the request is broadcast to web services located in the corresponding shard. If an id is registered to the web services metadata, a shallow copy of the corresponding object is loaded from the database servers. Finally the **XOBEDBPL** web service sends those shallow objects via SOAP as response to the client, which can either be a **XOBEDBPL** local server or another web service. The loading process is illustrated in figure 6.

Storing. The **XOBEDBPL** web service client sends the data object having to be kept persistently in connection with an identification id. An identification id can either be a session id, which is sufficient for simple transactions, or a transaction id, which is needed to perform complex transactions. After checking the identification id, the web service has to decide where to store or respectively update the object data. In case the data object is stored for the first time, our current heuristic is that the web service contacts its background database and stores the data locally. Otherwise the object data is stored on the corresponding original web service's host. Future strategies will move object data from one host to another, if it turns out that this host is more suitable, e.g. its directly

Actions:

1. send store request
2. check client's identification id and search metadata
- 3a. store new or local object on local host
- 3b. broadcast object update to registered XWS
4. receive and send response with stored data object ids

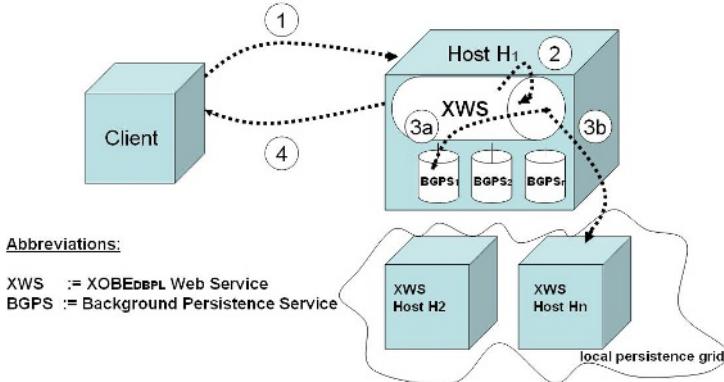


Fig. 7. Storing a data object with known type in single transaction mode

connected **XOBE_{DBPL}** local hosts work more frequently with this object data. So far we have assumed that the web service, which is going to store a new object, already knows how to do this, e.g. that he knows the structure or more precisely the type. Hence, if an object of a new type is stored, the web service's response asks for the corresponding type descriptors. In this case the web service's method called `registerType` can be used by the client to answer. Once again the client can either be another **XOBE_{DBPL}** web service or a **XOBE** local server instance. The storing process described so far can be seen in figure 7 in case of a known type and in figure 8 in case of a new type.

Deleting. The **XOBE_{DBPL}** web service client requests an object deletion by sending an identification id and a selection expression. After checking the identification id the web service evaluates the selected set of object ids analogously to the load case. With this information the web service forwards the delete request to its background database or respectively broadcasts it to the corresponding **XOBE_{DBPL}** web services being part of the shard. At the moment objects are deleted in a shallow manner, since referenced objects are supposed to be part of an aggregation. Additionally, objects, which are requested to be deleted are marked and physically deleted after a specific time intervall. This mechanism can be compared with a garbage collector in the Java programming language. The delete process is shown in figure 9.

Besides selection expressions, data objects and class descriptors, successful communication with the **XOBE_{DBPL}** web service enables session ids to identify clients and transaction ids to guarantee consistent access of distributed objects. Session ids as well as transaction ids can be requested from the **XOBE_{DBPL}** web service.

Ids in the whole **XOBE_{DBPL}** runtime environment are generated in a decentralized manner by using the SHA-256 hashing algorithm. I.e. the input value to calculate an object id consists of its creation time, its type and client information. Accordingly, the

Actions:

1. send store request
2. check client's identification id and search metadata
3. send unknown type response
4. send register type request
5. continue to store object
6. store object locally
7. send response with stored object id

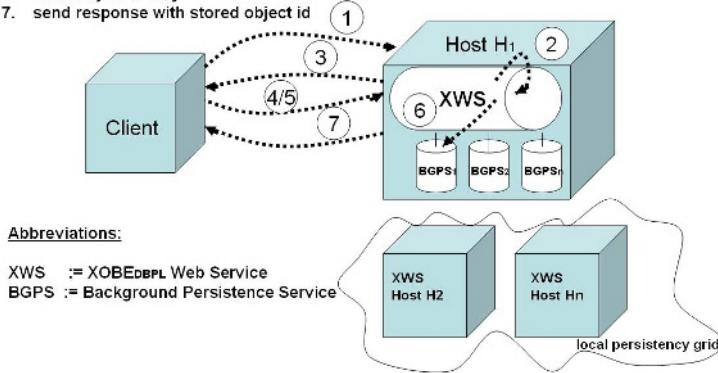


Fig. 8. Storing a new data object with unknown type in single transaction mode

input value for a session id consists of the client's IP address and a time stamp. The probability of a collision in case of object ids in $\text{XOBE}_{\text{DBPL}}$ is $1:2^{56}$, hence generated ids are statistically unambiguous. Another concept to generate unambiguous ids is that every host and client administrates its own counter. Ids are generated with the help of this counter and additional information, e.g. the unambiguous IP address. Further concepts to generate ids can be found e.g. in [23].

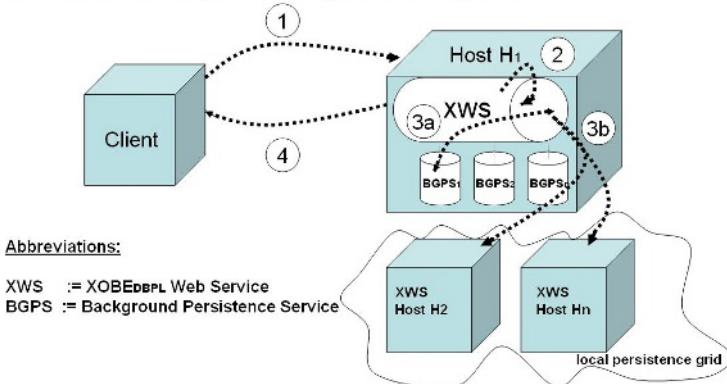
Lifecycle of a session id. A client requests a session id by passing client specific data. After checking this data the server sends a session id with an initial lifetime, which is rather short. Besides the web service generates its own customer id to identify the client. Further requests from the same client extend the lifetime of the corresponding session id. A client can finish any session by sending the id back to the $\text{XOBE}_{\text{DBPL}}$ web service.

Lifecycle of a transaction id. After receiving a valid session id, the client can perform load operations and store and delete operations in simple transaction mode. To perform modifying operations like store and delete in complex transaction mode, a transaction id is needed in addition. Transaction ids are given to a client on the basis of a session id.

Transactions. As mentioned before every single modifying operation which is requested with a session id is handled and executed as a single short transaction internally. Only the corresponding object is locked for a short time. Complex transactions, initialized in case of $\text{XOBE}_{\text{DBPL}}$ transaction blocks, which consist of more than one modifying operation, are requested along with a transaction id. These operations are queued by the corresponding directly connected web service. Finally, at the moment the client sends its request to execute the transaction, all involved objects are locked on request of the directly connected web service instance. The locking request can occur on

Actions:

1. send delete request
2. check client's identification id and search metadata for selected object ids
- 3a. delete local object
- 3b. broadcast delete request to registered XWS
4. receive and send response containing deleted data object ids

**Fig. 9.** Deleting a data object

different hosts by different web service instances. Finally and only if the original web service instance receives a positive result all operations are executed. At the moment **XOBE**_{DBPL} supports a standard flat transaction model based on a two phase commit protocol. Nested transactions are not supported yet.

Local and Shared Data Management Optimization. The **XOBE**_{DBPL} web service is completely integrated in the **XOBE**_{DBPL} runtime environment. Due to performance reasons and if persistent data is not going to be shared it can be kept locally as well. A running **XOBE**_{DBPL} program interacts with the **XOBE** local server as mentioned before. Its interface is designed analogously to the web service's interface. An instance can either connect to the web service or use a local persistency mechanism, e.g. a local database.

5 Application Example

This section describes an application scenario using **XOBE**_{DBPL}. We assume that a scientific institute wants to manage bibliography entries persistently. Bibliography entries are modeled according to the DBLP description as defined earlier in this paper. Each employee of the institute may create new bibliography entries. A new bibliography entry is supposed to be shared among all institute employees. Consequently an employee may work with all existing bibliography entries of the institute. A realization based on **XOBE**_{DBPL} is illustrated in figure 10. The institute contains a server network including three hosts (Host 1, Host 2, Host 3) and nine clients. Here, clients are the employees personal computers and hosts are servers on which the bibliography entries are stored. Furthermore each client possesses a **XOBE**_{DBPL} runtime environment executing the bibliography program of listing 2. A **XOBE** web service instance is deployed on each host. As explained previously a **XOBE** web service manages its own meta data and some persistent entries. The meta data contains the addresses of all **XOBE**

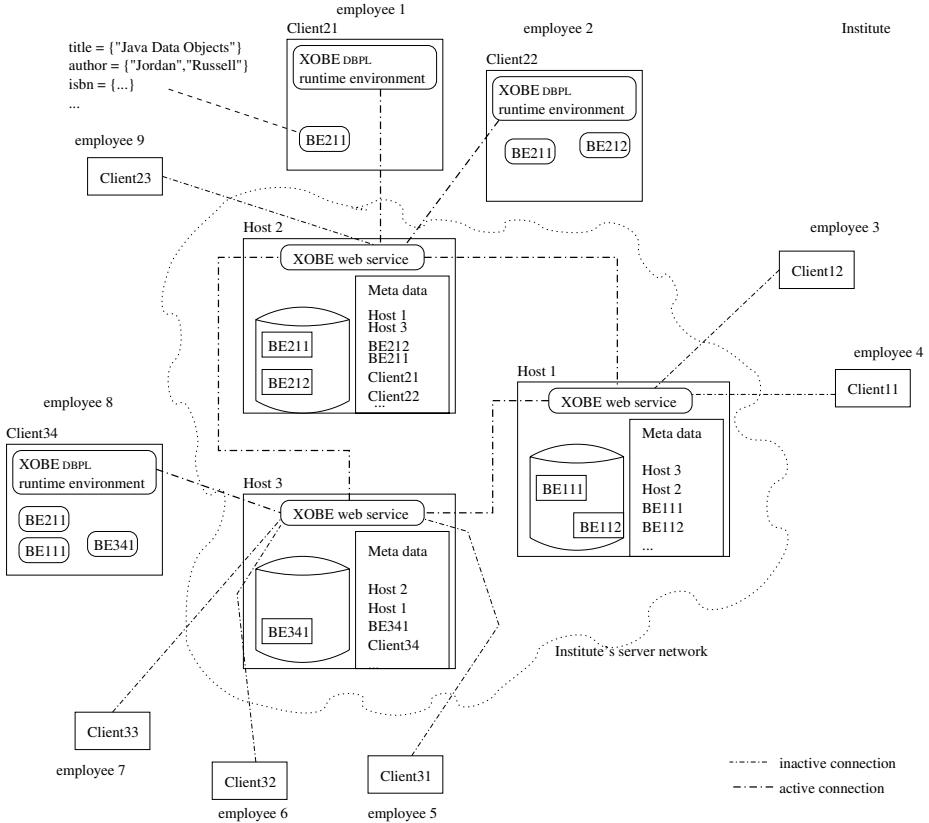


Fig. 10. Institute working with bibliography entries

web service hosts being part of the same network, ids of locally available bibliography entries and finally information about the currently connected XOBEDBPL clients. E.g. the meta data of Host 2 indicates that its directly connected web service instances are located on Host 1 and Host 3, the bibliography entries BE212 and BE211 are stored locally and Client21 and Client22 are registered at the moment. As one can see from figure 10, all three web service instances are connected with each other. A client's XOBEDBPL runtime environment is directly connected to a specific host's XOBEDBPL web service. In the institute a client is connected directly to its closest host's, e.g. Client21 is connected to Host2. Clients which are currently inactive because the bibliography program and its runtime environment are not executed, are drawn without inner details, e.g. Client12. Bibliography entries are abbreviated by BE followed by a number indicating the client which originally created the entry and an index number. I.e. the entry BE211 was created by Client21 and has 1 as index number. Figure 10 shows that entries are located on the host which is closest to the corresponding client, e.g. BE341 was created by the program running on client Client34 and is now stored on the host Host 3. Due to the realization of the web

service network in **XOBE_{DBPL}** (see section 4), clients can work with all bibliography entries regardless of location and origin, e.g. client Client34 is working with entries BE211, BE111 and BE341. Realizing this application example with **XOBE_{DBPL}** means profiting of distribution concepts and location transparency. Although entries are geographically distributed, the programmer can act as if all entries are in one central host. Another interesting aspect is the fact that most often each employee creates and works with topic specific entries. Since his own entries are stored on the closest host, the current **XOBE_{DBPL}** configuration runs at optimal cost. First experimental tests in which 1000 entries were stored and worked with gained constant and respectively less than linearly increasing times in the size of 10 ms.

6 Related Work

Distributed Objects. CORBA [1] is an OMG specification and a basis to develop distributed applications which can exchange messages independent of hardware, operating systems and programming languages. Besides incompatibility, it lacks of an own object oriented programming language interface. In contrast to CORBA, Java Remote Method Invocation (Java RMI) [25] was developed for a heterogenous Java environment. It does not need a new interface description language like CORBA since Java already includes interfaces. CORBA as well as Java RMI do not support persistency aspects explicitly, they both realize distributed object architectures.

Persistent Objects. The Java Data Objects specification (JDO) [2] reached final status in the year 2002. JDO consists of an interface specification between the application layer including the persistent capable entity objects and the persistency framework. Details about the persistency framework are not given or defined. Statements about used technologies, e.g. XML or relational databases, or realization details, e.g. mapping details are not made. Persistent capable classes must be defined and described in a separate XML file. The programmer has to connect to a central instance dealing with persistent objects. This persistent manager must be called explicitly each time a specific object should become persistent. JDO's persistent objects are not language independent. Hibernate [4] is an open-source object relational mapping framework for Java environments, meant to shield developers from most common data-persistency-related programming tasks. It also has a specific query language. Hibernate offers facilities for data retrieval and update, connection pooling, transaction management, declarative entity relationship management, and declarative and programmatic queries. In contrast to **XOBE_{DBPL}** the data model for storing data is not transparent and fixed to relations. Moreover the specific O/R mapping for persistent classes has to be defined by the programmer explicitly. In contrast to Java Spaces [5], a persistence technology loosely based on Tuple Spaces [26], **XOBE_{DBPL}** emphasizes full automation of persistence for objects. Java Spaces uses read, write and take commands to manipulate persistent data. These commands must be executed explicitly in order to make an entry persistent. **XOBE_{DBPL}** overcomes this necessity.

Distributed and Persistent Objects. Like JDO, Enterprise Java Beans (EJB) [3] and their client side counterparts Java Beans provide a framework for distributed persistent objects in Java. More recent versions include web service capabilities. They try to

reduce programming difficulties and exception overhead. Nevertheless EJB are not developed as a database programming language. Therefore, the EJB programmer is forced to make objects persistent and to connect to a central persistency instance explicitly by writing specific code for this task.

Database Programming Languages. To the best of our knowledge there is no database programming language which integrates the XML data model into an object oriented language like Java. The concept of a database programming language includes especially transparent and type independent persistency as well as a transaction concept. Among the approaches which integrate the relational model is DBPL [27] and its successor project Tycoon [28]. In the DBPL project Modula-2 is extended by parametric bulk types for relations. Its successor project Tycoon is based on an object-oriented programming language.

7 Concluding Remarks

In contrast to existing approaches **XOBE_{DBPL}** is a database programming language with transparency regarding type, persistency and distribution aspects of objects. Application development can be done exclusively on a high business object level. In this paper we introduced the concepts for realizing distributed persistent objects in **XOBE_{DBPL}**. A web service for distributed persistency and data exchange is used internally. Most importantly this web service is totally hidden from the **XOBE_{DBPL}** programmer and fully integrated in the **XOBE_{DBPL}** runtime environment. Furthermore we have described an application scenario managing scientific bibliography entries proving the feasibility of our approach. Future work will concentrate on transactions and communication times within the **XOBE_{DBPL}** runtime environment. Several starting points have been mentioned in the paper. In this context we will start to perform further tests comparing the **XOBE_{DBPL}** approach with other approaches introduced in the related work.

References

1. Object Management Group (OMG): Common Object Request Broker Architecture (CORBA). OMG CORBA, <http://www.corba.org/> (2005)
2. Sun Developer Network: Java Data Objects (JDO) . Sun Developer Network, <http://java.sun.com/products/jdo/> (2005)
3. Sun Developer Network: Enterprise JavaBeans Technology. Sun Developer Network, <http://java.sun.com/products/ejb/> (2005)
4. Hibernate: Hibernate. URL: <http://www.hibernate.org/4.html> (2005)
5. Sun Developer Network: Jini Network Technology - Specifications. Sun Developer Network, <http://java.sun.com/software/jini/specs/> (2005)
6. Kempa, M., Linnemann, V.: Type Checking in XOBE. In Weikum, G., Schöning, H., Rahm, E., eds.: Proceedings of Datenbanksysteme für Business, Technologie und Web (BTW), 10. GI-Fachtagung,. Volume P-26 of Lecture Notes in Informatics., Gesellschaft für Informatik (2003) 227–246
7. Kempa, M.: Programmierung von XML-basierten Anwendungen unter Berücksichtigung der Sprachbeschreibung. PhD thesis, Institut für Informationssysteme, Universität zu Lübeck (2003) Aka Verlag, Berlin, (in German).

8. Schuhart, H., Linnemann, V.: Updates for Persistent XML Objects. In Vossen, G., Leymann, F., Lockemann, P., Stucky, W., eds.: Proceedings of Datenbanksysteme für Business, Technologie und Web (BTW), 11. GI-Fachtagung,. Volume P-65 of Lecture Notes in Informatics., Gesellschaft für Informatik (2005) 245–264
9. Schuhart, H., Linnemann, V.: Implementing A Database Programming Language For XML Applications. In Guimaraes, N., Isaias, P., eds.: International Conference Applied Computing(IADIS). Volume 1., International Association for Development of the Information society (2005) 153–161
10. Schuhart, H., Pietzsch, D., Linnemann, V.: Framework of the Xobe Database Programming Language. In: International Conference Applied Computing(IADIS). (2005) 193–200
11. W3Consortium: XML Path Language (XPath), Version 2.0. W3C Working Draft, <http://www.w3.org/TR/xpath20> (2004)
12. W3Consortium: Extensible Markup Language (XML) 1.0 (Third Edition). Recommendation, <http://www.w3.org/TR/2004/REC-xml-20040204/> (2004)
13. W3Consortium: XML Schema Part 0: Primer Second Edition. Recommendation, <http://www.w3.org/TR/xmlschema-0/> (2004)
14. Ley, M.: Digital Bibliography and Library Project. <http://dblp.uni-trier.de/> (2005)
15. Poet Software GmbH: FastObjects. URL: <http://www.fastobjects.com> (2004)
16. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: Enabling scalable virtual organizations. Lecture Notes in Computer Science **2150** (2001) 1–25
17. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration (2002) Foster, I., Kesselman, C., Nick, J. and Tuecke, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Globus Project, 2002, www.globus.org/research/papers/ogsa.pdf.
18. java.net: Java Compiler Compiler (JavaCC) – The Java Parser Generator. <http://javacc.dev.java.net/> (2004) Version 4.0.
19. Apache XML Project, T.: Xerces Java Parser. <http://xml.apache.org/xerces2-j/> (2005) Version 2.7.1.
20. Group, U.C.: Java Tree Builder JTB. <http://compilers.cs.ucla.edu/jtb/> (2004) Version 1.3.2.
21. Apache Axis Project, T.: axis. <http://ws.apache.org/axis/index.html> (2004) Version 1.2.
22. PostgreSQL Global Development Group: PostgreSQL. <http://www.postgresql.org/> (2005)
23. Ambler, S.W.: Mapping Objects To Relational Databases. <http://www.AmbySoft.com/mappingObjects.pdf> (2000)
24. Beyer, K., Cochrane, R.J., Josifovski, V., Kleewein, J., Lapis, G., Lohman, G., Lyle, B., Özcan, F., Pirahesh, H., Seemann, N., Truong, T., der Linden, B.V., Vickery, B., Zhang, C.: System rx: one part relational, one part xml. In: SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM Press (2005) 347–358
25. Sun Developer Network: Java Remote Method Invocation (Java RMI) . Sun Developer Network, <http://java.sun.com/products/jdk/rmi/> (2005)
26. Gelernter, D.: Multiple tuple spaces in linda. In: PARLE (2). (1989) 20–27
27. Schmidt, J., Matthes, F.: The DBPL Project: Advances in Modular Database Programming. Volume 19 of Information Systems. (1994) 121–140
28. Matthes, F., Schröder, G., Schmidt, J.: Tycoon: A Scalable and Interoperable Persistent Environment. Fully Integrated Data Environments, ESPRIT Basic Research Series, Heidelberg, Springer-Verlag (2000) 365–381

Comparing Service-Oriented and Distributed Object Architectures

Seán Baker¹ and Simon Dobson²

¹ IONA Technologies plc, Dublin IE

sean.baker@iona.com

² School of Computer Science and Informatics, UCD Dublin IE

simon.dobson@ucd.ie

Abstract. Service-Oriented Architectures have been proposed as a replacement for the more established Distributed Object Architectures as a way of developing loosely-coupled distributed systems. While superficially similar, we argue that the two approaches exhibit a number of subtle differences that, taken together, lead to significant differences in terms of their large-scale software engineering properties such as the granularity of service, ease of composition and differentiation – properties that have a significant impact on the design and evolution of enterprise-scale systems. We further argue that some features of distributed objects are actually crucial to the integration tasks targeted by service-oriented architectures.

1 Introduction

Distributed Object Architectures (DOA) provide a stable computing platform on which to co-ordinate information systems within enterprises. The increasing desire to integrate very large, very loosely-coupled systems, and to automate business-to-business interactions, has led to an appreciation of the complexities of extending DOA technologies across enterprise boundaries. Service-Oriented Architectures (SOA) have been introduced with the goal of providing both intra- and inter-business services, and so potentially acts both as a complement to, and replacement for, DOA as an enterprise platform.

SOA and DOA are aimed at different problems, however. The origins of DOA lie with systems such as the CORBA standard, which was defined as a general-purpose integration technology. Because there was little support for implementing servers when CORBA was introduced in the early 1990s, CORBA implementations (and indeed the standard), concentrated on helping programmers to implement clients and servers. CORBA's commercial success made it a rival for other middleware: CORBA and the various other middlewares formed middleware islands, each with good internal integration, but poor inter-island integration. J2EE was introduced, among many other reasons, to provide a distributed object facility for Java.

SOA has been introduced to tackle the highest-level integration task, one important aspect of which is the integration of these middleware islands.

DOA can be very specific to one specification or standard; SOA has to be more technology neutral. SOA has to work at a wider scale across an enterprise, across middleware islands in an enterprise, and across enterprises themselves.

One may debate, therefore, whether SOA is a minor re-adaptation of DOA ideas to a more XML-based world, a new departure in middleware that differs in fundamental ways from what came before it, or a business-driven replacement for old-style technology-driven middleware designs that better reflects modern concerns. This paper contributes to the debate by analysing the features that make SOA and DOA similar and identifies a number of subtle features that make them different. We take a more architectural perspective than that of Vogels[1], while reaching many similar conclusions. In particular, we argue that some of the features that some SOA advocates deprecate – especially standardised interface types and fine-grained object decomposition – might actually be key features in the integration task that SOA is targeting, while other features that are sometimes cited as being core differences – such as the use of messages rather than method calls – are of little deep significance. We argue that, despite being small individually, the sum of these differences leads to a radically different set of properties for enterprise-level system modeling and design.

Section 2 highlights the superficial similarities between the two approaches and argues that these mask the deeper differences which are explored in more detail in section 3. Section 4 concludes with some observations on how the features sets of DOA and SOA point towards an more complete approach to enterprise integration.

2 Similarities

For most of this paper we use Web Services and CORBA as prototypical examples of SOA and DOA respectively. This is simply to provide a concrete context for discussion, and should not be taken as suggesting that either system is a pure exemplar of the architectural style. The choice of Web Services carries some particular worries: it is very new technology and therefore not yet mature, and as it matures it may begin to concentrate too strongly on its own specifications and lose sight of the broader, technology-neutral requirements of SOA. We believe that, because SOA is used at such a high level within enterprises, it cannot be based solely on any one middleware.

Perhaps unsurprisingly there is no universally agreed definition of either SOA or DOA: however, there is a certain convergence on the general thrust of the two architectures:

Service Oriented Architecture. “A service is a set of functionality provided by one entity for the use of others. It is invoked through a software interface but with no constraints on how the functionality is implemented by the providing entity . . . A service is opaque in that its implementation is hidden from the service consumer except for (1) the data model exposed through the published service interface and (2) any information included as metadata

to describe aspects of the service which are needed by service consumers to determine whether a given service is appropriate for the consumer's needs." (OASIS)

Distributed Objects Architecture. "[Distributed object] applications are composed of objects, individual units of running software that combine functionality and data, and that frequently (but not always) represent something in the real world. Typically, there are many instances of an object of a single type ... For each object type you define an interface. The interface is the syntax part of the contract that the server object offers to the clients that invoke it. Any client that wants to invoke an operation on the object must use this interface to specify the operation it wants to perform, and to marshal the arguments that it sends. When the invocation reaches the target object, the same interface definition is used there to unmarshal the arguments so that the object can perform the requested operation with them." (OMG)

These bare definitions obviously share many similarities, and one might argue (as indeed several commentators have) that SOA is simply a marketing re-invention of DOA. However, the similarities are actually rather deceptive, and we believe that some of the differences, although subtle, are vitally important in understanding the true relationship between the two approaches.

First the similarities. Both SOA and DOA are structured around remote entities (services or objects) which perform actions on behalf of clients. The remote entities typically export a strongly-typed interface defined using a language-neutral interface definition language (WSDL[2] or IDL[3]), transported using a language-neutral wire protocol (SOAP[4] or IIOP[3]). Not all implementations function in exactly this way: Java's J2EE architecture is based on distributed objects but has a language-specific interface language and wire protocol, although J2EE objects can also act as part of CORBA systems.

Perhaps the most common comparison between SOA and DOA contrasts SOA's technological neutrality against DOA's lack of flexibility. Web services are presented as being able to use lightweight HTTP interactions, but without being tied to HTTP or WSDL, and this is contrasted against CORBA's dependence on heavyweight IIOP and IDL. This comparison is largely specious, in principle if not always in practice. CORBA's interoperable object references (IORs) encapsulate multiple "profiles" for accessing the same object *via* different protocols, allowing an ORB that supports a number of protocols to access the object using whichever protocol is most appropriate. It would be perfectly possible for an ORB to use HTTP as an optimised lightweight transport while transparently interoperating with every other CORBA installation. Equally the dynamic stub and skeleton interfaces provide independence from IDL when required. (See [5] for more details.) The point here is not to praise CORBA unnecessarily but rather to focus the debate on actual rather than superficial differences between systems.

Both SOA and DOA provide invocation by clients of operations at remote sites. DOA "prefers" two-way interactions both because this matches the needs of a typical application and because CORBA poorly defines the semantics of one-way calls; and by default a method call invokes a remote operation and waits

for it to complete. SOA is more neutral, in that it supports interactions being constructed explicitly from messages. This makes one-way interactions simpler to construct.

Some authors have contended that this difference in invocation forms the essence of the difference between DOA and SOA. However, experience shows that this is deceptive. CORBA, for example, supports both synchronous and asynchronous communication. The former predominate, and normal use of IDL interfaces implies synchronous communication: however, one-way calls can be defined and more advanced RPC structures such as “promises” can be used to make two ways calls non-blocking[6]. By contrast, a large fraction of SOA systems make almost exclusive use of two-way synchronous calls constructed from messages. While DOA and SOA may have different “preferences”, the similarities dominate¹. While there *is* a difference between methods and messages it actually occurs elsewhere, not in the basic calling conventions – a point we return to in section 3.7.

SOA and DOA place similar emphasis on the signatures of operations, but place different emphases on interfaces types for the services themselves. DOA systems emphasise the use of interface compilers to construct client-side stubs for invoking methods. While some commentators have stressed that SOA operations can be called individually, without interface compilation, our experience in practice is that large-scale SOA systems are using interface compilers too.

Both SOA and DOA are essentially families of systems sharing common architectural principles, and it is instructive to see the extent to which these sets of principles also overlap. SOA advocates stress the separation of systems into services with well-defined interfaces accessed using a common communications framework, allowing composition of services and removing the boundaries between applications and middleware islands. DOA advocates would provide a remarkably similar list: indeed, CORBA’s initial *raison d’être* was to provide such a bridge between different applications, later extended to operate between middlewares. The point of both architectures is to provide *interoperability* rather than *homogeneity*.

3 Differences

What, then, *are* the differences between SOA and DOA? The (alleged) novelty and dynamism of the former are often set against the (alleged) rigidity of the latter: it is hard to see how this can be true given their clear technical similarities. The source of these comparisons may actually reflect the differences in maturity between the two approaches: DOA has a small number of dominant implementations, and the features and restrictions of these are used to define the overall approach. Even though SOA is too new now for such definition and restriction, the two have always have strong similarities.

However, perhaps the defining characteristic of enterprise-scale software architecture is its sensitivity to small differences, and we believe that SOA and

¹ It is certainly the case, however, that synchronous CORBA interactions are significantly more optimised than asynchronous calls in most ORBs.

DOA place this in high relief. Despite the similarities described above there *are* differences, and moreover *these differences count in aggregate*. A system engineered with SOA will be significantly different from one engineered with DOA, especially in terms of its long-term evolution. Moreover each approach has features that could be used beneficially by the other. It is these effects that we explore in this section.

3.1 Granularity

Granularity refers to the size of entities that are independently addressable within a system. In DOA systems these entities are individual objects; in SOA they are services. While this is a distinction that is, it must be admitted, impossibly subjective, we believe that some useful general observations may be made.

DOA inherits from standard (single-host) object-oriented programming a preference for fine-grained interfaces, although needing a somewhat coarser granularity to perform well over a network. Interface designers and programmers are encouraged to divide systems using interfaces providing a single abstraction (strong cohesion) and exhibit minimal dependence on the implementations of other interfaces (weak coupling). In designing a system for a travel agent, for example, a DOA designer might define interfaces for individual travel itineraries with methods for costing, booking and querying, and then make these accessible through an interface to a travel agent that collects together the itineraries and provides additional aggregate operations (figure 1(a)).

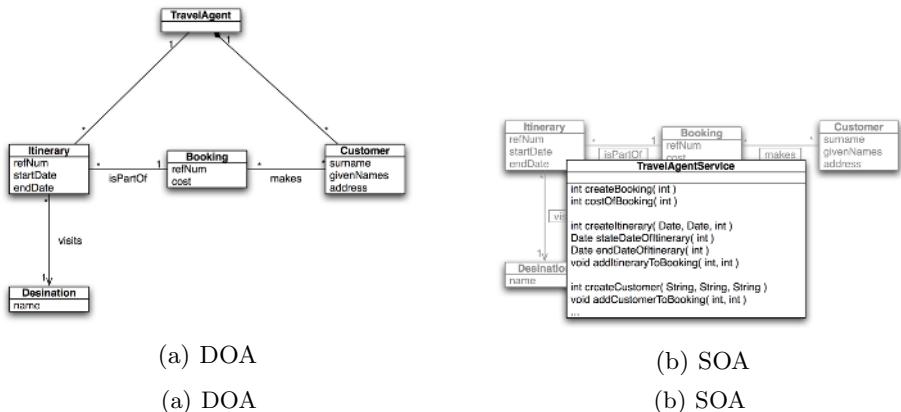


Fig. 1. Different granularities of visible objects

A SOA designer, by contrast, would probably take a more coarse-grained approach, for example designing a travel agent service providing operations on itineraries, with the itineraries themselves being specified by a reference number. The travel agent interface would provide itinerary creation, querying and so on, without exposing itineraries as an abstraction in their own right – although they would probably exist within the implementation (figure 1(b)). Indeed, the internal object model in the SOA case may be identical to that in the DOA case, differently exposed.

The difference is more than superficial. At a modeling level, DOA encourages an approach that is more purely object-oriented in the sense of identifying each abstraction and making it available in its own right as a first-class object. SOA encourages a decomposition in terms of real-world entities that have a concrete existence at the business level, and which captures the relationships and interactions between business entities without introducing other, extraneous abstractions.

As a SOA system evolves, new service interfaces will be introduced only when a new business entity is used. Continuing the example, a travel agent's interaction with a travel-insurance company will result in an insurance service interface being defined, again encapsulating the insurance business' business-level facilities. A DOA system evolves similarly, except that introducing an insurance service would also involve introducing a collection of smaller abstractions and interfaces (for policies, claims *etc*) that would typically be elided in public in the SOA case.

The coarse-grained, SOA view is that, from a business perspective, a travel agent is an entity "worthy" of an interface whilst itineraries are not: business interoperability should occur at the level of businesses, not with those businesses' internal abstractions. The finer-grained, DOA view is that itineraries, policies, claims *etc* are domain objects in their own right and should be modeled as such. This obviously reflects more than a simple difference in technology, and cuts to the heart of the differences in approaches promoted by the different architectural styles.

There are significant advantages to the SOA view. It reduces the "surface area" of interfaces, and hence the learning curve for client programmers. It also tends to produce interfaces that perform operations in fewer interactions. The DOA view may lead to interactions that are too "chatty", in the sense of requiring a number of interactions to accomplish the same effect. In many systems this will increase the number of network operations to accomplish a single business-level task, reducing system throughput.

This may also explain why concurrency control and transactions are a significantly more visible issue in DOA than in SOA. A fine-grained system will inevitably involve multiple object interactions in a single business task, and so will need to make transactions visible to clients. SOA by contrast exposes the business tasks explicitly, and so is better able to abstract concurrency control behind the service interface. However the SOA approach needs concurrency control in order to scale, and these issues are re-appearing at the services level to support the emerging activity on web services transactions.

A significant literature has grown up around design patterns for object-oriented systems, some part of which now targets distributed object systems. (Good examples are [7, 8]). One immediate observation is that these design patterns almost always result in finer-grained decompositions of objects. This suggests that the usual object-oriented approaches translate well to DOA (with the caveat that they must be balanced against the need to avoid over-chatty interactions in the interests of performance and robustness), but not necessarily so readily into SOA.

3.2 Interface Types

DOA systems typically place significant emphasis on the definition of interface types. Interface types appear for applications (for example, standardised objects models for different vertical domains), for common services (for example the CORBA services and facilities definitions), and in the standards for DOA middleware themselves. The use of interfaces is pervasive and can be used to great effect: the CORBA trading service provides an interface for choosing between instances of an interface, with the interface repository providing a machine-readable description of the interface types.

SOA places less emphasis on common interface types: some commentators actually go further, asserting that SOA services should not have interface types that can have multiple instances[9]. However it seems inconceivable that a widely-deployed SOA infrastructure would not converge, at least to some extent, at least *ad hoc*, within specific vertical markets, on a set of commonly-agreed interface types. For example, a travel agency consortium could specify a shared definition and insist that all of its members adopt it. It is not sensible to prevent the consortium defining such an interface type – as indeed many are already – and the notion of SOA would be weakened if it does not actively support this.

Standardising interface types requires that there is an authority able to manage the standard. This authority may be defined in system-centric terms (all users of this system agree to these interfaces), or may be defined per-vertical (a set of common interfaces for the travel industry) or globally (a common interface to discovery services). In DOA there is typically strong agreement throughout a given system – although the system may be very large and most programmers may only be familiar with the interfaces of the sub-systems with which they work. In SOA by contrast – which targets even larger systems than DOA, and which specifically deals with systems spanning enterprises – there is no global attempt to control interface types.

Standardised interfaces within industry verticals can provide leverage at the modeling level as well. If we again consider the example from above, a travel agent using a travel insurance company will often be able to re-use the existing, agreed object model for insurance companies, reducing both the costs of extension and the degree of coupling between parties.

SOA advocates argue that the lack of standardised interface types increases flexibility and dynamism. However, interface standardisation evolved for a reason: without such interfaces, system developers must explicitly write adapters for each component. As system size and complexity increase, creating and (much more importantly) maintaining this “glue” code becomes the dominant cost. It is impossible to avoid the concern that large-scale SOA deployment may be better able than DOA to encompass a wider range of disparate systems, but only at the cost of hugely increasing the amount of interface adaptation and maintenance required.

In many large systems each interaction will typically only use a small fraction of each interface. This is perhaps more true for SOA than DOA, as the interfaces involved tend to be larger (section 3.1). One might argue that SOA adapters can

therefore focus on providing only the part of the interface required, which may simplify their development. Experience suggests that this argument is specious over the long term, as increasing complexity will tend to “fill out” the use of interfaces over time.

The reduced emphasis on interface type standards therefore does not remove the pain of standardisation for enterprises, but instead simply defers it to integration time and replicates it for each integration.

3.3 Composition

The usual response to an overly fine-grained decomposition (typically manifested as poor performance) is to re-engineer the system to coalesce several interfaces.

In many cases the internal design of a system will follow its external decomposition, with objects for itineraries *etc*. Coalescing interfaces will typically take the form of providing a façade that hides the individual objects.

Composition of services is an integral part of SOA, often referred to as *orchestration* and supported by a number of emerging standards (for example BPEL4WS[10]). Individual “partners” in a composite service are specified by providing the service port, optionally including a port type (interface type). Many programming languages have similar constructions, notably the structure and signature system in Standard ML[11].

One can provide orchestration without interface types. However, the absence of standardised interfaces make it difficult to see how a single process description can be re-used on different instances of services. The point is that interfaces provide more than individual operation signatures. An interface combines a set of operations with an implicit (or, increasingly, an explicit) contract on the way in which these operations will work together. Adherence to an agreed, reviewed and documented standard provides developers with confidence – albeit sometimes misplaced – that the operations will function together as intended and will respect the interface’s contract. Allowing orchestration on the basis of individual operations, without this level of confidence in their consistent underlying assumptions, seems unlikely to succeed on a large scale.

3.4 Identifying Instances

Both SOA and DOA provide names for instances of interfaces: SOA typically identifies services by URLs for their service endpoints where queries should be directed, providing direct integration with the web, while DOA systems typically use more opaque identifiers such as CORBA IORs or J2EE object references.

A fine-grained DOA decomposition means that individual entities will typically have a distinct identity. We can identify an itinerary using an object reference, and both interact with it directly and pass it to other objects for them to use (figure 1(a)). By contrast a SOA decomposition focuses on service endpoints

which will not typically identify such a small object with a (SOA-level) identifier, and so will use an *ad hoc* reference such as itinerary number, reflecting the submerging of the object model behind the business interface.

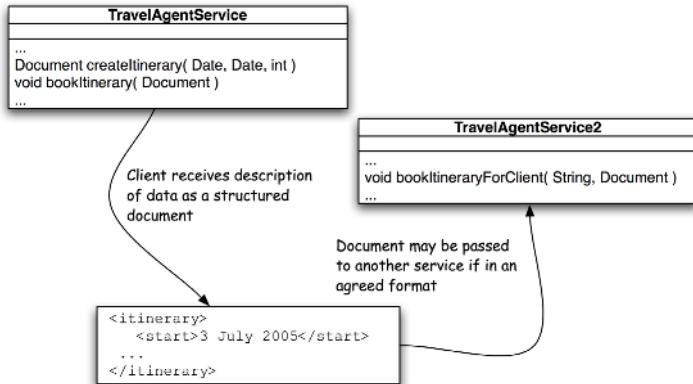


Fig. 2. Standardised documents replace standardised interfaces for data exchange

The coarse-grained approach has an immediate impact on systems architecture. Since data objects cannot be referred to directly by reference, the designer has two options. The first alternative is to identify internal data by some form of reference number, forcing the service implementor to manage an *ad hoc* namespace of objects. These namespaces are private, in the sense that an identifier generated by one instance cannot be used by another instance. To use the travel agent example again, a client must keep track of which travel agent service is managing which itinerary, so that queries are directed to the right interface. A service that used multiple services, for example when booking hotels through multiple providers, would be faced with quite a complex task. In SOA these issues must be managed outside the framework; in DOA they are typically managed by the middleware. This weakens SOA's claim to provide location transparency[12]: while technically true, the added complexity of namespace management reduces its impact in practice.

The second alternative (figure 2) is to manage data using a more document-centric approach, with a service creating documents that describe the data being manipulated: the travel agent provides an itinerary document in XML to the client, which may then store it or pass it back to the service at a later point. Such documents can be passed between services *if* they have an agreed format, so some level of standardisation across enterprises is again needed to facilitate integration. Services still need to convert documents to and from an internal representation and must take steps against alteration or forgery of data held externally, both of which complicate data management and interface coding.

If SOA is widely deployed, private namespaces and/or document formats will inevitably proliferate and complicate interactions. It seems possible to us that the DOA model therefore has significant advantages in terms of orchestration, service composition and re-purposing, as the single notion of object provides a significant simplification at the level of composition even though it may complicate individual interactions.

Some more structured approaches to endpoint description are emerging. BPEL4WS encourages the instantiation of business processes to handle stateful interactions, using correlation sets to identify specific instances[10–section 10]. WS-Addressing also provides more structure for addressing into individual services[13]. Neither approach provides support for the service implementor in managing their internal namespaces or entity lifecycles.

3.5 Modeling

Is the SOA or DOA modeling approach better? The answer essentially depends on the view we take as to what is happening when an analyst or modeler produces an object model. Both views may be derived depending on our initial assumptions:

If we take the view that the set of interface types identified are fundamentally tied to the system being modeled, capturing its essence, then it follows that these are unlikely to change unless there is a fundamental change made to the system. If such a fundamental change *is* made, any model – wherever it lies on the spectrum of granularity – will have to change. It also follows that such a set of interfaces will be easy to use because they model the real world so well. It *also* follows that making each object visible will allow improved re-use and reduce complexity, since the designer can focus on providing only the different functions in the system at each interface. We would therefore conclude that exposing the detailed object model will improve and simplify the long-term evolution of the system.

On the other hand, if we take the view that the object model is a detail in the day-to-day internal running of a business, it follows that the model will be subject to frequent change as the business evolves. It thus follows that such change should be masked from clients as it has no real significance to business-to-business interactions; would lead to undue coupling between businesses; and would anyway be too complicated for an external client to understand as understanding the decomposition (and hence the functions available) would involve too much understanding of the business. (This is essentially a model-level version of the well-known “yo-yo problem”[14] at the code level.) We would therefore conclude that exposing only business-level functions, preferably in one interface and with no visible dependencies on other services, will improve the robustness of business-to-business interactions in the face of evolution.

Both positions are perfectly defensible. Our own view is that SOA cannot be dogmatic on the approach used. The guidance from practice would be that mainstream object-oriented decomposition can lead to over-fine interfaces for DOA. Interfaces cannot be so fine-grained that they will perform well only over very

fast networks: this would lead to too tight a coupling between a service and its clients. However, the SOA principles themselves – and certainly the technology used to implement them – should not restrict the modeling approach that an enterprise believes in. In particular, it is too fine a point for an architecture to regulate on how granular a set of interfaces should be.

Objects can be business level or system level. Viewed simplistically, services must be business level: however, it is difficult to draw the dividing line between business- and system-level entities, and this raises much the same issues around modeling as above. In addition, the principles behind SOA (such as well-defined contracts and separating implementation and interface) are just as valid at the system level as they are at the business level. (Some commentators even use the term SOA for the system level, and the term B-SOA for the business level; we prefer to define SOA as the uppermost level of integration and use the term Enterprise Service Bus to refer to the technology that facilitates communication between SOA entities).

In both SOA and DOA, it is common for programmers to generate service/object interfaces from some form of interface on existing systems (perhaps a description of the data that is exchanged, or interfaces defined in programming languages). These lead to low-level – and often inappropriate – interfaces. The only excuse for doing this in DOA is as a stepping stone to offering a proper high level interface to clients; there is probably no excuse for doing it in SOA.

The rise in interest in model-driven architecture[15] provides a possible bridge between the modeling views. MDA defines a system architecture as a set of transformations between models to reflect the different levels of concern in systems development. Coarse-grained SOA interfaces fit well into this scheme, although this still leaves the problems (exemplified in section 3.4) that arise from the inability to share model objects across enterprises.

3.6 Inheritance and Implementation

Inheritance is often regarded as a core characteristic of object-oriented systems [16]. Many DOA systems support inheritance at two distinct but related levels. At the interface level, an interface may specialise another by adding new operations. At the implementation level, it is often possible (although not required) to re-use the implementation of one interface in defining the specialisations of that interface.

Not all DOA systems have this feature, however. J2EE is particularly restricted in this regard, in that there is exactly one implementation of each interface within a single application instance, and that implementation is (somewhat perversely) not a valid Java-level instance of its own external interface type. The lack of polymorphism means that many approaches used in other DOA systems do not translate well into J2EE. It also complicates some simple but highly effective optimisations, most notably using language-level references instead of external network references when exchanging references to objects within the same server.

WSDL does not provide the notion of interface specialisation, perhaps in keeping with SOA’s reduced emphasis on interface types altogether². However, sub-interfaces provide a vital mechanism for differentiating providers without compromising interoperability: a client may use additional features provided by a particular provider if it understands them, but may rely on a core of operations being available uniformly regardless of provider. Run-time interface manipulations and sub-interface checks make this process reasonably straightforward. We persist with our view that interface types are actually *more* important for SOA than for DOA, given the focus on cross-enterprise integration.

Inheritance is not the only form of extension, however. Delegation-based models are also extremely powerful, and neither SOA nor DOA typically support the notion intrinsically. While such mechanisms are useful linguistically, they can be approximately provided using events (see below).

3.7 Operations, Messages and Events

Returning to the use of messages *versus* operations, one may ask the question *why* this difference is so insignificant, given the widespread belief that messages (in one form or another) are an essential feature of loosely-coupled systems.

Both SOA and DOA allow clients to invoke operations on individual interfaces. While messages (or one-way methods) may decouple the request from its completion, they do not change this model.

By contrast, event-based systems allow information to be transferred to zero, one or more consumers according to several possible models. In the CORBA event service’s “push” model, receipt of an event automatically triggers the execution of handler code, so a single event can cause code to be invoked on the *different* objects that are subscribed to the *same* event channel.

The key difference is not (as is sometimes stated) between messages and method calls as invocation mechanisms, but rather between first-class events which may be manipulated programmatically and second-class invocation mechanisms that operate beneath the language level. To use the terminology of aspect-oriented programming[17], events reify the calling mechanism into the language and allow it to be modified to (for example) queue invocations or distribute them more widely. The XML nature of SOAP messages makes them easier to manipulate programmatically. Such manipulations are also common in CORBA systems however, albeit with considerably more effort being required.

Both SOA and DOA infrastructures have external event services (often with a standardised interface type in the case of DOA). Clients may interact with such event services using either messages or method calls, which are then converted into events and operated on using filters, publish/subscribe *etc*. While events provide powerful support for loose coupling, SOA’s explicitly message-based invocation mechanism provides little or no such advantage over DOA.

² Although sub-interfaces may be added to later versions.

4 Conclusion

We have explored the similarities and differences between the service-oriented and distributed object approaches to enterprise system modeling, design and implementation. While the approaches (and their underlying technologies) share significant features in common, their differences – while subtle – lead to significantly different views of enterprise system structuring and evolution. This strongly suggests that SOA in particular represents more than simply a marketing phenomenon, and conversely that DOA has both a significant on-going niche and a number of important lessons to impart.

Of all the issues examined, four stand out. Firstly, SOA requires a significantly coarser granularity of exposed object model than DOA. The focus on coarse-grained, aggregated interfaces may simplify interactions across enterprises (or business divisions) by reducing the number of interface interactions needing to be understood. This is an important simplification in a world in which businesses interact more dynamically and without necessarily establishing long-term relationships between their information systems: programmers need to understand less of the target businesses' infrastructure in order to avail of its services.

Secondly, the ability to exchange references to objects within the framework seems to be a positive step for orchestration, allowing different providers to work with each others' data directly rather than *via* private references or descriptive documents – which require standardisation anyway. Forcing designers to deal with these additional complexities seems to serve no useful purpose.

Thirdly, SOA's de-emphasising of interface types will not lead to simpler integration over the long term. While simpler to establish, such *ad hoc* approaches push complexity out into each integration, the costs of which will spiral as SOA increases in market penetration. It would seem prudent to repeat the OMG's experience in defining standardised object models and interfaces within vertical market segments, allowing providers to differentiate themselves in other ways or by providing extensions to the basic structures.

Finally, business integration did not start with SOA. There is a existing volume of work in modeling the issues, objects, operations and relationships of specific industries. The significance of this work lies in the understanding it gives of industries and the degree of commonality that exists between providers, rather than in the object models *per se*. Perhaps, then, a more appropriate, business-level question is: how can this understanding be re-used in the slightly different context of SOA? It may be that the underlying modeling assumptions are too different to allow simple translations, although providing a service-oriented façade might allow SOA to use the existing DOA infrastructures with little additional cost.

These are extremely subtle issues, but taken together the different solutions that SOA and DOA take to them aggregate to deliver distinctively different system architectures. SOA systems are likely to use more asynchronous invocations and to involve less up-front cost to establish, although the complexities of adaptation may over time make this a rather Pyrrhic victory. DOA systems may

more closely reflect a conceptual object model of the application domain and make information sharing and exchange simpler, but only in situations where standard object models and interface definitions can be agreed upon and when the value of the expected interactions justifies the standardisation costs.

The messages *versus* remote method calls debate misses the point. The distinction is rather between first- and second-class invocation mechanisms, and flexible event systems are equally definable in – and complementary to – both SOA and DOA, and so do not provide a reason to choose one over the other. Indeed, Gartner Group have suggested that *any* system architecture requires both invocation-based and event-based interactions to maximise loose coupling.

SOA's goals of a more “business-friendly” distributed platform are laudable, economically significant and technologically challenging. It is however important to remember and re-use the features from more traditional DOA approaches that can usefully be included into the top-level integration of enterprise systems.

References

1. Vogels, W.: Web services are not distributed objects. *IEEE Internet Computing* **7** (2003) 59–66
2. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: *Web Services Description Language 1.1*. Technical report, World Wide Web Consortium (2001)
3. Henning, M., Vinoski, S.: *Advanced CORBA programming with C++*. Addison Wesley (1999)
4. Winer, D.: XML-RPC specification. <http://www.xmlrpc.com/spec> (1999)
5. : Common Object Request Broker Architecture (CORBA/IOP), v.3.0.3. Technical Report formal/2004-03-12, Object Management Group (2004)
6. Liskov, B., Shrira, L.: Promises: linguistic support for efficient asynchronous procedure calls in distributed systems. In: Proceedings of the ACM SIGPLAN conference on Programming Language Design and Implementation, PLDI'88, ACM Press (1988) 260–267
7. Mowbray, T., Malveau, R.: *CORBA design patterns*. Wiley (1997)
8. Fowler, M.: *Patterns of enterprise application architecture*. Addison Wesley (2003)
9. Webber, J.: Horses for courses: services, object and loose coupling. *Web Services Journal* **4** (2004)
10. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services, version 1.1. Technical report, IBM (2003)
11. Milner, R., Tofte, M., Harper, R., MacQueen, D.: *The definition of Standard ML (revised)*. MIT Press (1997)
12. Waldo, J., Wyant, G., Wolrath, A., Kendall, S.: A note on distributed computing. In Vitek, J., Tschudin, C., eds.: *Mobile object systems: towards the programmable Internet*. Springer-Verlag (1997) 49–64
13. Box, D., Christensen, E., Curbera, F., Ferguson, D., Frey, J., Hadley, M., Kaler, C., Langworthy, D., Leymann, F., Lovering, B., Lucco, S., Millet, S., Mukhi, N., Nottingham, M., Orchard, D., Shewchuk, J., Sindambiwe, E., Storey, T., Weerawarana, S., Winkler, S.: *Web services addressing (WS-Addressing)*. W3C member submission (2004)

14. Taenzer, D., Ganti, M., , Podar, S.: Object-oriented software reuse: the yoyo problem. *Journal of Object-Oriented Programming* **2** (1989) 30–35
15. Frankel, D.: Model driven architecture: applying MDA to enterprise computing. Wiley (2003)
16. Cardelli, L., Wegner, P.: On understanding types, data abstraction and polymorphism. *ACM Computing Surveys* (1985) 471–522
17. Kiczales, G., des Rivières, J., Bobrow, D.: The art of the metaobject protocol. MIT Press (1991)

QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms

Michael C. Jaeger, Gero Mühl, and Sebastian Golze

Techn. Universität Berlin, Institute of Telecommunication Systems,
Sek. FR6-10, Franklinstrasse 28/29, D-10587 Berlin, Germany
`{golze, gmuehl}@ivs.tu-berlin.de, mcj@cs.tu-berlin.de`

Abstract. A composition arranges available services resulting in a defined flow of executions. Before the composition is carried out, a discovery service identifies candidate services. Then, a selection process chooses the optimal candidates. This paper discusses how the selection can consider different Quality-of-Service (QoS) categories as selection criteria to select the most suitable candidates for the composition. If more than one category is used for optimisation, a multi-dimensional optimisation problem arises which results in an exponential computation effort for computing an optimal solution. We explain the problem and point out similarities to other combinatorial problems – the knapsack problem and the resource constraint project scheduling problem (RCPSP). Based on this discussion, we describe possible heuristics for these problems and evaluate their efficiency when used for web service candidate selection.

1 Introduction

A Web service composition is a collection of single Web services that form a new, more complex service. The creation process of a composition can be divided into several phases. In the first phase, the control flow – the execution arrangement of individual tasks – is defined. The idea is that available Web services realise individual tasks needed for the composition. Different so called *flow languages* (e.g. BPEL4WS) are already available to describe such a flow. Based on the flow description, a discovery service identifies suitable task candidates. Then, a selection process chooses the optimal candidate for each task. After the assignment of available Web services to the tasks has been fixed and saved in a flow description, an execution engine uses the flow description to execute the composition. An execution engine can also provide the composition as a new *composed* Web service that is ready for invocation by third parties.

In the Web service domain, an open initiative hosted by the OASIS group proposes a specification for the service discovery called *Universal Description, Discovery and Integration*, in short UDDI [13]. The UDDI approach is part of the Web Services Architecture promoted by the W3C [2]. Another, more technology independent view on a service architecture is given by the ISO Reference Model for Open Distributed Processing (RM-ODP). In the RM-ODP, a so called *trader* facilitates the discovery and the selection process [7]. A trader matches requirements of service requesters to

services that are offered by service providers. First, functional requirements are the relevant criteria for a matchmaking process in the service discovery which results in a set of candidates. In addition, non functional requirements represent preference criteria. A selection process can involve such criteria to identify the optimal service.

According to this separation proposed by the RM-ODP, this paper will explain how a trader *selects* Web services by different optimisation criteria for compositions comprised by more than one Web services. Thus, it is presumed that a preceding discovery process has identified a set of candidates that matches the functional requirements for the referring tasks. If a trader has to select an individual Web service from the set of candidates, a trading function identifies the most suitable service by comparing the discovered candidates. However, in a composition a trader should identify the optimal set of Web services which are about to be combined. Selecting for each tasks the best candidate in isolation will in most cases not lead to an optimal solution. This paper explains why this problem can be regarded as a combinatorial problem and discusses possible solutions. For our discussion, we consider different QoS criteria that represent non-functional characteristics that we can quantify using different metrics. First, we give an overview about QoS for Web services and explain how the QoS of individual services can be aggregated in compositions. After the problem description and the introduction of possible solutions, the related work is discussed. The paper ends with our conclusions and future plans in this field of research.

2 QoS in Web Service Compositions

In this paper, we use different QoS categories to define requirements, which serve as selection criteria for available services. Each QoS category has an increasing or a decreasing direction. An *increasing* direction means that a higher value indicates a better quality, for *decreasing* categories vice versa. A set of QoS categories has been already introduced by various authors for the use in workflows by Cardoso [3] or in the Web service domain by Zeng et al. [17], or Menasce [10]. From these categories we have chosen the following four to give a more concrete discussion and examples in the remainder of this paper. Please note that our methods and algorithms will work also for other QoS categories:

Maximal Execution Time (Decreasing). The execution time defines the amount of time to execute the service. Different definitions are possible, which may include different phases of the invocation of the Web service. In this paper it is presumed, that covering the values of individual services will result in the overall execution time of the composition. In other words, delays or interrupts of the control flow are ignored.

Cost (Decreasing). The cost defines generally the amount of resources needed to use a service. Like execution time this measure is decreasing meaning that a lower value is preferred.

Reputation (Increasing). The concept of a reputation is basically about a ranking given by users of the service. For example, the auction platform eBay allows clients to rank the behaviour of other clients [17]. The reputation is defined as the average of the individual ranks of users.

Availability (Increasing). The availability denotes the probability that the execution of the node performs successfully and is regarded as an increasing dimension.

For the description the QoS of Web services different proposals already exist. Tosic et al. have proposed the Web Service Offerings Language (WSOL) [12] which covers among other issues also the definition of QoS statements covering a Web service. WSOL represents an XML-based language and has the focus on specifying the non-functional aspects of Web service. WSOL directly builds upon a WSDL description. Considering the WSOL as well as other languages, we need to point out that the definition of the used QoS concepts must be given individually. For example, WSOL covers this issue with an external reference to a common definition of the particular QoS category.

2.1 QoS Aggregation

In order to select candidates for tasks of a composition that is based QoS of the individual services, a method is needed to calculate the resulting QoS of the whole composition. In a preceding paper we have introduced an aggregation method to calculate the QoS of the composition based on the QoS of the individual services [8]. The model identifies seven basic structural elements called *composition patterns*. These structural elements were derived from a set of workflow patterns by van der Aalst et al. [15]. Workflow patterns form a set of functional and structural requirements for workflow management systems.

Among different reasons, we have chosen to use the workflow patterns because van der Aalst has shown that the structural part of the workflow patterns also applies to commonly known flow languages for compositions [14]. Thus, we can assume that our elements cover these flow languages as well. Since the scope of this paper does not allow to explain which of the workflow patterns we have considered as a composition element, we would like to refer the reader to the according analysis in our preceding paper [8]. From this analysis, we identified the following composition patterns:

Sequence of service executions. A sequence can either prescribe a specific order in which the services have to be executed or the services can be executed in an arbitrary order. For the aggregation model, the order of the executions is not relevant.

Loop. The execution of a service or a composition of services is repeated for a certain amount of times.

XOR split followed by an XOR join. From a parallel arrangement only one task is started and the synchronising operation waits for this started task.

AND split followed by an AND join. From a parallel arrangement all tasks are started, and all tasks are required to finish for synchronisation.

AND split followed by a m -out-of- n join. From a parallel arrangement all n tasks are started, but $m < n$ tasks are required to finish for synchronisation.

OR split followed by OR join. From a parallel arrangement a subset of the available tasks are started, and all of the started tasks are required to finish for synchronisation.

OR split followed by a m-out-of-n join. From a parallel arrangement a subset of n tasks from the available are started, and $m < n$ tasks are required to finish for synchronisation.

Using these elements, we aggregate the QoS for each category and for each pattern element. Figure 1 shows the pattern-wise aggregation of a simple composition example. To aggregate the maximum execution time, an algorithm would start to determine the largest value in the parallel sub-arrangement. Then, the sum of the sequential arrangement including the aggregated value of the parallel arrangement is calculated. This approach enables us to view the aggregation in a pattern-perspective, i.e. not the whole composition is relevant at once, but rather the local pattern elements. Following this approach, we have defined aggregation rules for each QoS category and for each composition element. Of course, further, more specific patterns are possible. The intention of the patterns is to deliver a model that allows to define a QoS statement that covers the composition. Such a statement will always represent an approximation of the delivered QoS during run time. Our model represents also an approximation and is subject to the trade-off between complexity and feasibility. We have described the patterns and the rules for different QoS categories in detail and compared them to other approaches in our preceding papers [8] [9].

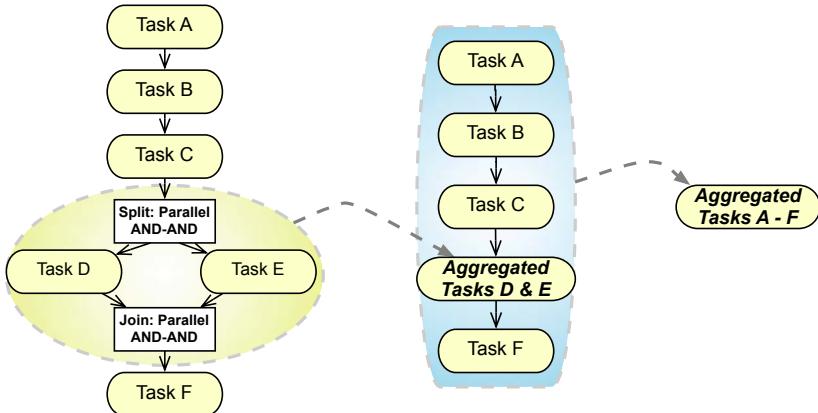


Fig. 1. Collapsing the Graph for Aggregation Example

2.2 The Selection of Services

The goal of the selection is to identify the best assignment of service candidates to the tasks of the composition. The selection can be performed by either considering or ignoring the arrangement of the tasks. When the structure is ignored, the selection can be formulated based on the following model:

Let the composition be a set of n tasks $\mathbb{T} = \{t_1, t_2, \dots, t_n\}$. The output of the previously performed discovery process is a set of m candidates $\mathbb{S} = \{\bar{s}_1, \bar{s}_2, \dots, \bar{s}_m\}$.

Each of the p QoS categories is assigned a unique number $1, \dots, p$. Then, each candidate can be expressed by a vector – a *QoS-vector* – with values $s_y, y \in \{1, \dots, p\}$ that represent the QoS categories that have been taken into account:

$$\bar{s}_x = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_p \end{pmatrix}$$

with $x \in \{1, \dots, m\}$ denoting the number of the according candidate. The selection process assigns one candidate to each of the tasks. Optionally, a selection could also identify two or more candidates for each task to increase the dependability of the tasks by executing all these candidates. However, we ignore this specific aspect for our discussion.

To find the optimal assignment, an algorithm must evaluate different combinations from a global perspective, i.e. the assignment cannot take place by considering each task and its candidates separately. A simple example clarifies the problem: consider the parallel arrangement of the two tasks D and E shown in Figure 1. Let the optimisation goal be to identify the composition which results in the fastest execution while trying to keep the lowest price. Also, a faster service is usually more costly – in other words, cost and execution time form a trade-off couple. In the case that even the fastest candidate for task D executes longer than any of the candidates for task E , the optimal assignment for the task E is the cheapest candidate, regardless how long its execution would take.

Thus for parallel arrangements, all combinations of assignments must be tested to find the optimal assignment. If a composition contains a parallel arrangement at the top level, all combinations for assigning candidates to the included tasks must be tested. The resulting effort increases exponentially with the number of tasks if for each tasks more than one candidate could be chosen: if the number of candidates increases by one, then the number of combinations to evaluate is doubled. If the number of tasks increases by one, the number of combinations increases multiplied by the number of candidates found for this task. Thus, for large numbers of tasks and candidates a global approach for the selection is not feasible and efficient heuristics are desired.

2.3 The Selection Criteria

The selection of candidates must be performed by applying some criteria. For the selection of candidates, particular QoS categories can be the basis for defining an optimisation function or optimisation constraints:

- One or more QoS categories are relevant for optimisation. Thus, for each considered QoS category the referring value of the QoS vector is subject to an optimisation function. This function depends on the direction of the dimension. For categories with a decreasing dimension (such as execution time or cost) the function is about to minimise the aggregated value, for categories with an increasing dimension it is about to maximise the value. The optimisation criterion is about to find the optimal – minimal or maximal – resulting QoS value aggregated by function f :

$$\{min|max\}(f(\mathbb{S}_y)) \quad \mathbb{S}_y = \{s_{1_y}p_1, \dots, s_{b_y}p_b\}$$

$$\text{with } p_x = \begin{cases} 1 & \text{if selected, and} \\ 0 & \text{otherwise.} \end{cases}$$

The index $y \in \{1, \dots, q\}$ refers to the considered QoS category.

- One or more QoS categories are relevant for expressing a constraint on the composition. Depending on the direction of the considered QoS category with index y , the constraint denotes an upper or lower bound for the resulting aggregated value:

$$\{c_y > |c_y <\}(f(\mathbb{S}_y)) \quad \mathbb{S}_y = \{s_{1_y}q_1, \dots, s_{m_y}q_m\}$$

$$\text{with } q_x = \begin{cases} 1 & \text{if selected, and} \\ 0 & \text{otherwise.} \end{cases}$$

3 Analogies to Knapsack and RCPSP

If exactly one QoS category is relevant for the optimisation, a selection algorithm must choose the candidate that offers the optimal referring value for each task. The effort for this operation is linear to the number of candidates and thus this specialisation of the selection problem can be regarded trivial. If more than one QoS category is relevant for optimisation or for expressing a constraint, the selection can be regarded as a combinatorial problem. This problem has similarities with the 0/1-Knapsack problem and to a specific kind of project scheduling problem (PSP). Our goal is to explain their differences to the selection problem and evaluate efficient (heuristic) solutions.

3.1 The 0/1-Knapsack Problem

The knapsack problem is about selecting a subset of available items for putting them into a knapsack. Each item has a specific weight, a specific value, and the knapsack has a limited weight capability. The problem is that the weight capability of the knapsack does not allow to take all items. Thus, a selection must be performed with the goal to identify the optimal subset which maximises the value while keeping the weight constraint. Transferred to the composition scenario, the knapsack problem could be formulated as follows:

- The composition represents the knapsack.
- Each candidate represents one item that can be put into the knapsack.
- A QoS constraint represents the limited weight capability of the knapsack.
- A QoS category which is subject to the optimisation represents the value of the item.
- The algorithm tries to find the optimal selection according to the optimisation function while keeping the constraint.
- Only "complete" candidates can be selected, a candidate cannot be split to meet the constraint. The knapsack problem, which does not allow to split items, is also known as 0/1-knapsack problem.

However, some characteristics of the selection differ from the known knapsack problem. The selection algorithm must also find a solution that (1) optimises the value of a given QoS category, (2) keeps the constraint and (3) selects a candidate for each task. For a normal knapsack, the solution might result in as few items as possible to keep the constraint.

If more than one QoS category is relevant for optimisation, the relevant values of the QoS vector of a candidate could be aggregated to form a new measure representing the value of the candidate. Then, the *Simple Additive Weighting (SAW)* can be applied, which was introduced in the context of *Multiple Criteria Decision Making (MCDM)* [6]. By applying this procedure, we can normalise the individual values and assign a score to each QoS vector. As first step, each value s_{yx} , with $y \in \{1, \dots, p\}$ indicating the QoS category and $x \in \{1, \dots, m\}$ indicating a particular candidate is replaced by the normalised value n_{yx} :

$$n_{yx} = \begin{cases} \frac{\max_y(s_{yx}) - s_{yx}}{\max_y(s_{yx}) - \min_y(s_{yx})} & \text{for decreasing categories} \\ \frac{s_{yx} - \min_y(s_{yx})}{\max_y(s_{yx}) - \min_y(s_{yx})} & \text{for increasing categories} \end{cases}$$

The result of this replacement is that the value representing the best quality results in a value of 1 and the worst results in a value of 0. Other values will range between 0 and 1. Then, a score c_x can be applied to each candidate [6]:

$$c_x = \frac{1}{p} \sum_{y=1}^p w_y n_{yx}$$

The weight w_y is applied to the QoS categories by the user's preference. The sum of all weights must be equal to 1. The result of this procedure is a score for each QoS vector representing a Web service candidate of an aggregated statement. Considering the knapsack problem, the overall score represents the quotient of value and weight of an item, because the score combines increasing and decreasing categories.

3.2 The Project Scheduling Problem

If multiple QoS categories are subject to the optimisation or if they form a constraint, the selection problem is similar to a so called *Resource Constrained Project Scheduling Problem (RCPSP)*. A project scheduling problem occurs when resources (usually humans) must be distributed to jobs of a project. The most common optimisation goal of the basic RCPSP is to reduce the duration of the project while spending as few resources as possible. Two types of RCPSPs are distinguished: one is called *Single Mode RCPSP* and the other is known as *Multi-Mode RCPSP (MRCPSP)*. The single mode RCPSP only deals with fixed values for the duration and the cost of a task. In a MRCPSP, a job can be done by using different modes which vary in cost and duration. Thus, a MRCPSP is considered for our selection problem. In addition to its mode, a RCPSP can be classified by a couple of other characteristics. Based on an overview about RCPSPs by Yang et al. the selection problem seen as RCPSP has the following characteristics [16]:

Objective. Objectives are distinguished by being regular or irregular. *Regular* objectives do not interfere with the goal to minimise the duration of the project, while *irregular* objectives allow to follow another objective – for example to equalise the consumed resources among involved parties [16]. Applied to the selection problem, the RCPSP has a non-regular objective because depending on the considered QoS categories and the applied weight, a worse duration can be considered as a better solution if, for example, the cost is reduced accordingly. The objective can be defined as an optimisation function as given in section 2.3.

Precedence Relation. Two types of precedence relations are discussed in the literature. Either tasks can be started with a specified time window after a preceding task has finished or a succeeding task can start any time after the preceding task has finished. Regarding a composition of Web services, the common case is that a task is started immediately after the preceding task finished.

The constraint for the precedence in Web service compositions can be defined as follows: let α_{x+1} be the start time of a candidate $\bar{s}_{x+1} \in \mathbb{S}_i$ of task i , $i = 1, \dots, n$, and ω_x the finish time of a candidate $\bar{s}_x \in \mathbb{S}_{i-1}$ for the preceding task $i - 1$. Then, the precedence constraint is:

$$\alpha_{x+1} q_{x+1} \geq \omega_x q_x \quad \text{with } q_{x+1}, q_x \left\{ \begin{array}{ll} 1 & \text{if selected} \\ 0 & \text{otherwise} \end{array} \right.$$

Between two tasks a time delay might occur, because an execution environment cannot execute a task at exactly the same moment another task has finished. However, for defining the constraint this interval is irrelevant and thus ignored.

Preemption. If preemption is allowed, the execution of a task can be suspended in order to execute another task. This is useful if some resources are only available within a specific period of time and other tasks can be suspended then. Since the invocation of Web services in the context of a composition is usually an atomic operation, preemption is not considered to be possible.

Resource requirements per period. In the domain of project scheduling, using resources at different time periods can result in different costs. For example, performing a task at night results in higher payments for night shifts. For computer systems a similar idea might be applied: the execution of a service gets more expensive at peak hours. Since we are not aware of any example that applies this idea we ignore this aspect in the following.

Trade-offs. In the context of RCPSPs, the trade-off is characterised by two criteria, where an optimisation of the one means a change to the worse for the other. An algorithm must find an counterbalanced solution. A usual trade-off pair is formed by time and cost in which case the cost and the execution time should be kept as low as possible. For the selection problem possible trade-off couples can be formed by cost vs. one of the other three mentioned categories in the sense that a higher quality results in a higher cost. However, a trade-off couple could be also time vs. availability thinking about a provider, where services usually execute very quickly but may fail quite often.

4 Approaches for the Selection Problem

Building onto the analogies among the two combinatorial problems explained in the previous section, our approach is to apply heuristics for these problems that are known to be efficient to the selection problem. In this section, four approaches are explained and compared:

Greedy Selection. For a greedy selection, the score c_x represents the selection criteria. The algorithm starts with calculating the score for each candidate. Then, to each task the candidate with the highest value density is assigned. It should be noted that if this approach is used, it is not possible to consider a global constraint.

Discarding Subsets. This algorithm represents a backtracking approach. It starts by parsing a search tree which consists of nodes each representing a possible pair of candidate and task. Each level of the tree holds pairs of a particular task only, resulting in the tree having the same number of levels as tasks. Each possible combination of candidate assignments is represented by a particular path of the tree from the root to a leaf.

To lower efforts, the algorithm cuts subregions, if (a) it can be determined that the constraint cannot be met anymore by following this particular subregion or if (b) we can guess that combinations represented by a particular subregion do not result in a better overall QoS than already found. If the algorithm finds an appropriate combination, it aggregates the overall QoS. The combination is stored if no combination has been identified so far resulting in a better QoS. The algorithm stops when all possible combinations have been evaluated. This approach will find a solution meeting the constraint, if it exists. However, it does not save any efforts in the worst case when worse branches cannot be identified at an early stage.

Since this approach normally identifies the optimal solution, it cannot be regarded a heuristic. We establish a cutting rule based on an estimation. Considering the execution time, the cutting rule is clear: if a complete combination has already been determined that shows a lower execution time as the partial combination processed at some moment, the algorithm cuts the subtree. Each additional candidate would worsen execution time. However, for categories where the aggregation calculates the arithmetic mean, a rule cannot determine whether the QoS gets worse or better. In our discussion, we consider the reputation as a QoS category which represents this case. Applied to the selection problem in our configuration, the discarding subsets algorithm guesses that the QoS gets worse and thus might ignore the optimal solution.

Bottom-Up Approximation. A large number of heuristics are already available for RCPSPs [16]. However, not every approach can be applied to the selection problem: RCPSPs and Web service compositions cover the execution order of tasks differently: for compositions the order is in most cases pre-defined in a flow description, while the tasks of a project are subject to precedence relations, which may allow to push a particular task for- or backwards in order to optimise the utilisation of resources. It turned out that several approaches for solving RCPSPs work

on a precedence model which does not allow the application to the selection problem. The resulting problem is that a solution space is created which considers the rearrangement of activities. The resulting bounding rules cannot be applied efficiently for the selection problem.

However, we have identified one heuristic covering a RCPSP introduced by Yang et al. which can be used for the selection [16]: The approach presumes that constraint and optimisation criteria form a trade-off couple, i.e. the quicker a task is performed the more it will cost. The heuristic applied to the selection would perform as follows:

1. The candidates are sorted by the QoS value s_{x_y} , with $x \in \{1, \dots, m\}$ and $y \in \{1, \dots, p\}$ denoting the QoS category for the constraint.
2. For each task, the candidate \bar{s}_x with the best s_{x_y} is assigned. If a solution exists, it is found with this step.
3. As the next step, the algorithm replaces the firstly assigned candidates by the candidate with the next worse value s_{x_y} .
4. The new combination is tested for whether the constraint is still kept. If the constraint is still kept, the algorithm continues by looping back to step 3. The algorithm stops, if at one time for each task no additional candidate is found that lets the composition meeting the constraint and increases the overall QoS.

Pattern-wise Selection. In a preceding paper we have introduced another heuristic [4], which directly covers the example of tasks *A* and *B* explained in section 2.2. We apologise that we cannot go into much detail about the pattern-wise selection due to spatial limitations. Thus, we refer to the mentioned publication for more information about its motivation and how it works. In very brief words, the algorithm determines the best assignment considering each composition pattern in isolation. The algorithm takes advantage of already identified elements of composition patterns (cf. Section 2.1). It performs four steps:

1. The algorithm walks recursively into the structure and identifies pattern elements that do not contain any sub-patterns.
2. For all tasks within this element, all sets of candidate assignments are evaluated. The QoS is aggregated for each combination by using the pattern-based aggregation. The combination that delivers the best score using the SAW procedure is chosen.
3. If the optimal solution for a particular pattern is determined, the algorithm walks one level upwards to evaluate the assignment within the new pattern. The aggregated QoS of contained sub-patterns is taken as a fixed value.
4. The pattern wise optimisation and aggregation is performed until the whole composition is covered and one aggregated QoS is returned.

Since this algorithm operates on each pattern element, this approach cannot meet global constraints. Thus, the pattern-wise selection is only suitable for optimising the overall QoS.

4.1 Comparison

We have compared the four proposed and two additional selection methods using a simulation environment: the additional methods are *a)* the *global* selection and the *b) constraint optimised* selection. The global selection evaluates all possible combinations and determines the best QoS possible for the composition. In addition to the resulting best possible QoS, the algorithm shows the worst case computation effort. By the second method, the candidates are sorted by the QoS category relevant for the constraint. Then, for each task the candidate offering the best QoS constraint category is assigned. Thus, if a combination which respects the constraint exists, it is found using this approach. However, this algorithm does not optimise other QoS categories and thus results in a poor QoS for the overall composition. The software simulation performs the following steps:

1. Generation of an arbitrary test composition structure by randomly arranging the composition pattern elements which are introduced in Section 2.1.
2. Generation of candidate Web services each with random QoS values for execution time, cost, reputation, and availability.
3. Performing each of the selection methods on the same composition structure with the same set of candidates.

We have tested the algorithms with arbitrary compositions with an increasing number of tasks (from 4 to 12) but a fixed number of candidates for each task (5). The implementation of the software and the random generation of candidates and their QoS values involve a large number of issues, which cannot be discussed completely in this paper due to limited space. Some issues among them are:

- Each test case with a particular number of tasks has been repeated 50 times with each time a new random composition structures, new optimal QoS values for each task and new resulting QoS values of the candidates. The results shown are the arithmetic mean values of the 50 repetitions for one test case. For each task the same number of candidates is generated.
- The simulator generates random composition structures by choosing one of the seven elements with equal probability.
- For each task an optimal QoS is randomly set and the randomly generated QoS values of the candidates are within 0 and 100 percent worse compared to the optimal value. This ensures a realistic distribution of the QoS values among the candidates referring to one task.
- To form a trade-off couple between execution time and cost, the two are set as follows: the percentage a added to the optimal execution time is taken to calculate the percentage b added to the optimal cost with $a+b = 100$. Thus, the more optimal the execution time is, the worse will be the cost and vice versa.
- The constraint is determined to perform the constraint selection on the cost first. The aggregated cost for the composition is increased by 20% and then taken as the constraint that has to be met by the other selection methods.

The results of this simulation are shown in Figure 2 and 3. Please note that in both figures the discrete results are connected with interpolated lines for better visualisation only. Figure 2 shows the average resulting QoS for the composition performing

the selection methods for the generated test compositions relative to the global selection, which always finds the optimal solution. For example, a QoS ratio of 0,80 means that this selection method has only gained 80% of the best QoS possible. The different aggregated QoS values resulting for a composition can be compared by applying the SAW approach to compute a normalised score (cf. section 3.1). As expected, the resulting QoS by using all other selection methods is worse than the global selection. The worst resulting QoS is delivered by the constraint selection, which does not optimise regarding the overall QoS.

The bottom-up method shows the next worse results. It results in a worse QoS than the greedy selection. However, the bottom-up method has met the given constraints while the greedy selection did not. Compared to the greedy selection, the pattern approach shows even better results. Among the methods which optimise the QoS while meeting a constraint, the discarding subsets methods shows the best resulting QoS. The results show that the average values seem to oscillate between a corridor. In fact, this corridor is quite small: for example for the difference between the highest and lowest average value of resulting QoS for the bottom-up method is lower than 0.02022 or less than 3% of the overall values. However, we need to admit that considering the wave-similar shape gives us the impression that future measurements should be repeated more often.

The discarding subsets method seems to increase its resulting QoS with a raising number of tasks whereas the greedy algorithms seem to get worse. This leads to the assumption that the discarding subsets algorithm copes better with larger/more complex structures. Vice versa, the declining performance of the greedy algorithm leads to the idea that this methods performs worse with more complex compositions.

Figure 3 shows the average execution times of the different selection methods for compositions with increasing number of tasks. Please note that the figure uses a logarithmic scale on its y-axis. The greedy selection and the bottom-up approximation show an almost linear increase of efforts with a larger number of tasks. Along with the simple constraint oriented selection, all three methods form a group of similar execution time behaviour: all calculations took about 1 millisecond or less covering compositions of from three to ten tasks. The bottom-up approach shows a break-out when testing for compositions with 4 and 6 tasks. Although the test computer has had a clean system installation, no connection to a network and the simulation application ran as the only user application at that time. Clearly, further activities must investigate this effect which might indicate an inefficiency of this algorithm depending on the number of tasks. But, since the measurements show that all results are still less than 1 millisecond, we regard the overall impact of this unexpected result as non-critical. The selection methods to determine the best QoS, discarding subsets and the pattern-wise selection form another group of similar execution time behaviour. All three methods scale exponentially, while the pattern-wise selection climbs slower with increasing number of tasks.

As explained in the previous section, it must be noted that the results from selecting the best QoS, the pattern-wise and the greedy selection do not necessarily meet the given constraint. The constraint limit for the cost is set to 20% worse than the optimal cost determined by the constraint-oriented selection. The three selection methods showed

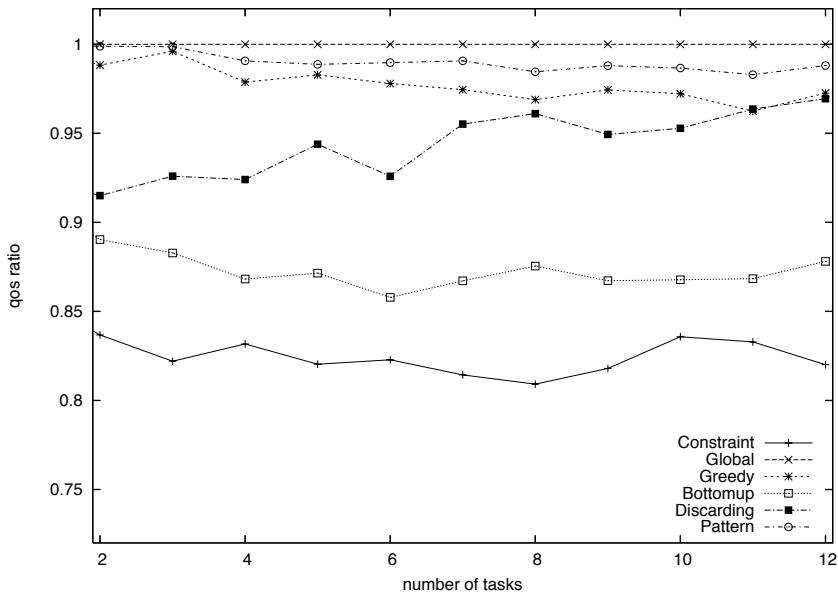


Fig. 2. Average Delivered QoS of Different Selection Methods Relative to Best QoS

together that the cost constraint is met incidentally by one third of the test cases with no noticeable relation to an increasing number of tasks. However, due to space limitations the results cannot be discussed in this paper.

5 Related Work

The work of Puschner and Schedel about calculating the execution time for software architectures represents our foundation for the aggregation of QoS in Web service compositions [11]. They have defined calculation rules for structural patterns as found in software executions. Since our composition model plays in the field of Web service compositions, we have adopted this principle and build our patterns onto the workflow patterns by van der Aalst et al. [15]. Cardoso has applied a similar approach in his work to calculate the QoS for workflows [3]. In his work he has identified different QoS criteria and defined calculation rules for these criteria in workflows based on the flow structures as found in the meteor workflow management system. Since our composition patterns are based on workflow patterns, we can also support structures like the two OR-split patterns along with the *m-out-of-n-join* constructs. As a consequence the composition patterns can be applied to more specific applications more easily.

Different authors have also discussed which QoS categories might be considered in Web service compositions [10] [17]. Their contribution has been taken up to determine the relevant categories for our work. If needed, the chosen categories can be extended without affecting the discussed approaches itself.

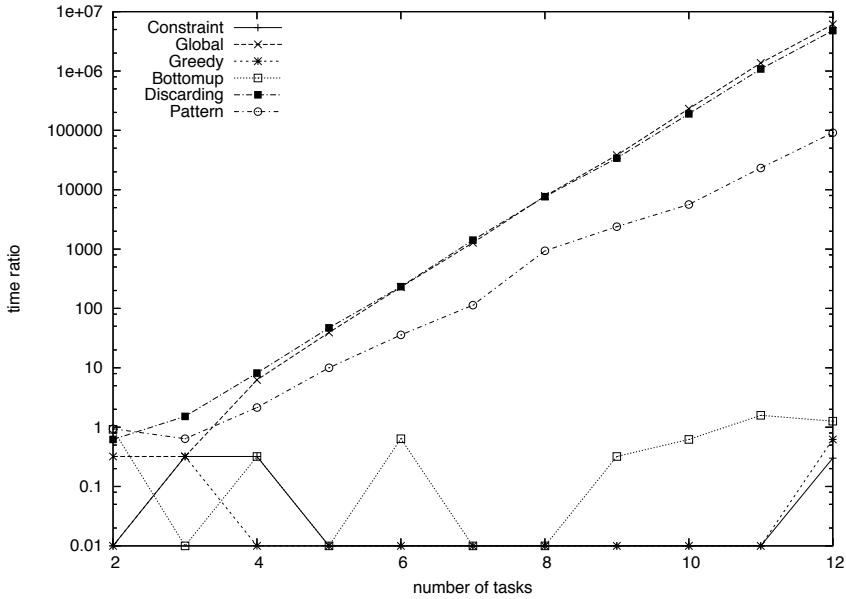


Fig. 3. Average Execution Times of Selection Methods

Using QoS statements as the main criteria for the selection of Web services is an already known idea, which has been introduced in fundamental papers at the time, when the Web services were emerging and the concept of Web service composition was presented [1] [5].

For the selection of candidates, Zeng et al. have identified two basic approaches: a local and a global selection [17]. To address the problem that a global selection has exponential effort, Zeng et al. have introduced an Integer Programming (IP) solution to compute the optimal assignment of services. Using the IP-approach reduces significantly the number of combinations by identifying constraints. According to their tests, the IP-based algorithm scales better with a growing number of candidates and tasks. We plan to test the IP-approach with our simulation tool to deliver a statement on how well it compares to pattern selection. Apart from the selection, Zeng et al. propose different aggregation mechanisms for their selection of QoS categories. We think that because of its uniform structure, the pattern-wise aggregation results in lower efforts for their implementation and the computation of the aggregation.

6 Conclusions

We have discussed different algorithms for performing the selection of candidates to optimise the overall QoS of a composition. Considering a computation algorithm, the used set of QoS categories can be extended or reduced depending on what is considered to be relevant for a specific application case. The intention of our work is to focus on the selection algorithms independent from the characteristics of different QoS categories.

For the optimisation *without* the need to meet a constraint, the results have shown, that a pattern-wise selection results in unfeasible efforts with a growing number of tasks. However, the pattern-wise selection performs still significantly quicker than the approach by discarding subsets. The pattern-wise selection also reaches almost the level of the best possible QoS. If a slight decrease of the overall QoS is tolerable (about 5% according to our results), the greedy selection delivers acceptable results with neglectable efforts. For the optimisation *with* the need to meet a global constraint our results have shown that the bottom-up approximation results in an overall QoS about 10% worse than for example the discarding subsets approach. However the computational effort of the bottom-up approximation remains feasible also with a growing number of candidates. The results show that for a selection in a time-critical scenario with a larger number of tasks, heuristics can be successfully applied with the penalty of a decrease on the overall QoS. Such an application scenario could be a reconfiguration of the composition during run-time when a Web service has become unavailable.

For future research in this direction possible test cases could operate on a fixed composition structure with an increasing number of candidates to evaluate the different algorithms with specific arrangements. Also the generation of the candidates and their QoS values could be improved in order to deliver more realistic example compositions. We also plan to evaluate other approaches such as genetic algorithms to solve the selection problem.

Acknowledgement

The authors would like to thank specifically the reviewer who did an excellent job by pointing out very good improvements and provided to us a very detailed list of suggestions.

References

1. Boualem Benatallah, Marlon Dumas, Marie-Christine Fauvet, and Fethi A. Rabhi. Towards Patterns of Web Services Composition. Technical Report UNSW-CSE-TR-0111, University of New South Wales, 2001.
2. David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture. <http://www.w3c.org/TR/ws-arch/>, February 2004.
3. Jorge Cardoso. *Quality of Service and Semantic Composition of Workflows*. PhD thesis, Department of Computer Science, University of Georgia, Athens, GA (USA), 2002.
4. Roy Gronmo and Michael C. Jaeger. Model-Driven Methodology for Building QoS-Optimised Web Service Compositions. In *Proceedings of the 5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'05)*, pages 68–82, Athens, Greece, May 2005. Springer Press.
5. Richard Hull, Michael Benedikt, Vassilis Christophides, and Jianwan Su. E-Services: A Look Behind the Curtain. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'03)*, San Diego, USA, June 2003. ACM Press.

6. Ching-Lai Hwang and K. Paul Yoon, editors. *Multiple Attribute Decision Making: Methods and Applications*, volume 186 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, March 1981.
7. ISO/IEC. ITU.TS Recommendation X.950 — ISO/IEC 13235-1: Trading Function: Specification, August 1997.
8. Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Mühl. QoS Aggregation for Service Composition using Workflow Patterns. In *Proceedings of the 8th International Enterprise Distributed Object Computing Conference (EDOC'04)*, pages 149–159, Monterey, California, September 2004. IEEE Press.
9. Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Mühl. QoS Aggregation in Web Service Compositions. In *The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pages 181–185, Hong Kong, China, March 2005. IEEE Press.
10. Daniel A. Menasce. QoS Issues in Web Services. In *IEEE Internet Computing*, pages 72–75. IEEE Press, November-December 2002.
11. Peter Puschner and Anton Schedl. Computing Maximum Task Execution Times - A Graph-Based Approach. *Journal of Real-Time Systems*, 13(1):67–91, July 1997.
12. Vladimir Totic, Kruti Patel, and Bernard Pagurek. WSOL – Web Service Offerings Language. In *Proceedings of the Workshop on Web Services, e-Business, and the Semantic Web - WES (at CAiSE'02)*, volume 2512 of *Lecture Notes in Computer Science*, pages 57–67, Toronto, Canada, May 2002. Springer-Verlag.
13. UDDI Spec Technical Committee. UDDI Version 3.0.1. <http://uddi.org/pubs/uddi-v3.0.1-20031014.pdf>, 2003.
14. Wil M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. *Jan/Feb 2003 Issue of IEEE Intelligent Systems*, pages 72–76, January 2003.
15. Wil M.P. van der Aalst and Arthur H.M. ter Hofstede and B. Kiepuszewski and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases* 14(3), pages 5–51, 2003.
16. Bibo Yang, Joseph Geunes, and William J. O'Brien. Resource Constrained Project Scheduling; Past Work and New Directions. Technical Report Research Report 2001-6, Department of Industrial and Systems Engineering, University of Florida, 2001.
17. Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Transactions*, 30(5):311–327, May 2004.

Extending the UMIOP Specification for Reliable Multicast in CORBA^{*}

Alysson Neves Bessani¹, Joni da Silva Fraga¹, and Lau Cheuk Lung²

¹ DAS - Departamento de Automação e Sistemas,
UFSC - Universidade Federal de Santa Catarina,
Florianópolis - Santa Catarina - Brazil
`{neves, fraga}@das.ufsc.br`

² Graduate Program in Applied Computer Science,
Pontifical Catholic University of Paraná, Curitiba - Paraná - Brazil
`lau@ppgia.pucpr.br`

Abstract. OMG has published an unreliable multicast specification for distributed applications developed in CORBA (UMIOP). This mechanism can be implemented based on IP Multicast, a best-effort protocol, which provides no guarantees about the message delivery. However, many fault-tolerant or groupware applications demand more restrictive agreement and ordering guarantees (for instance, reliable multicast with FIFO, causal or total ordering) from the available support for group communication. OMG has not yet provided any specification for supporting those requirements. This paper presents an important contribution towards this direction. We proposed the ReMIO, an extension to the UMIOP/OMG protocol, for the conception of a reliable multicast mechanism in CORBA middleware. Performance measures comparing ReMIO, UMIOP and UDP sockets for IP multicast communication are presented in order to evidence the costs for adding reliable and unreliable multicast in middleware level.

1 Introduction

When CORBA architecture (*Common Object Request Broker*) [20] was introduced by OMG (*Object Management Group*), only point-to-point communications (using static or dynamic invocation) was available through the ORB (*Object Request Broker*). The messages that pass through this channel obey a proper transference syntax defined by the GIOP (*General Inter-ORB Protocol*). This syntax makes the messages involved in the communications independent from ORBs implementations and the consequences of an heterogeneous environment. The mapping of GIOP over TCP/IP transport layer is made by IIOP (*Internet Inter-ORB Protocol*) protocol. The IIOP and TCP/IP combination is a good solution for distributed objects communications in the client/server model, since it considers aspects like error control, FIFO ordering, etc.

* This work is supported by CNPq (Brazilian National Research Council) through processes 401802/2003-5 and 481523/2004-9.

Point-to-point communications have shown, in general, effective in distributed applications supported by CORBA. However, many of these applications would have a better performance, concerning time, memory and message complexity, if they could use multi-point communication mechanisms. Usually, these applications depend on abstractions like groups of objects or the need to disseminate data over several hosts of a network. Therefore, group oriented applications could have greater benefits from the network low-level services.

In an attempt to supply the need of multi-point communications in CORBA middleware-level, OMG published the UMIOP (*Unreliable multicast Inter-ORB Protocol*) specifications [19] in 2001. The UMIOP is a set of specification for an unreliable multicast service to be included as part of the ORB. The protocol defined in these specifications, the MIOP (*Multicast Inter-ORB Protocol*), is responsible by GIOP mapping over UDP/IP multicast stack. IP multicast is a set of IP protocol extensions that enables it to establish multi-point communications [7,6]. This protocol is characterized by absence of guarantees and high performance, mainly in local networks. Many applications use IP multicast, especially those for distributed multimedia systems.

The unreliable multicast service defined by UMIOP, the less restrictive group communication model, can be used for some distributed applications, for example, video conference, in which the loss of some frames do not represent the degradation of transmitted information. However, fault-tolerant applications, groupware applications, among others, usually demand more restrictive guarantees concerning group communication supports reliability and ordering (for example FIFO, causal, total, etc). OMG has not yet provided any specification concerning these requirements. This problem is being treated by OMG in stages. The first step, therefore, was the publication of UMIOP specifications. We believe that initiative motivates OMG to publish another RFP (*Request for Proposal*), towards a Reliable and Ordered Multicast Inter-ORB Protocol. Initial submissions to this RFP has been already done [21].

The integration and implementation of UMIOP in an ORB were presented in [1,3]. As a step forward, this paper presents our contributions in the conception of a “best-effort” reliable multicast support in the ORB, based on UMIOP specifications. The proposed model, called ReMIOP, is CORBA and UMIOP specifications compliant - the proposed extension does not change any interfaces of the current specification. Actually, we indicate how to integrate reliable multicast protocols into ORB without any change of the CORBA specifications. The inclusion of a reliable multicast protocol on top of the MIOP layer is implemented as a plugin mechanism. Some performance measures of the ReMIOP (the ReMIOP/MIOP/UDP/IP multicast stack), UMIOP (MIOP/UDP/IP multicast stack) and UDP sockets (UDP/IP multicast stack) are presented to show the costs of including reliable and unreliable multicast in middleware level.

This work is part of the GROUPPAC project [16,2], which is a set of object services based on FT-CORBA specification (chapter 23 of [20]) and developed to make easier the implementation of fault-tolerant distributed applications.

This paper is organized as follow: section 2 presents the OMG initiatives for group communication introduction in CORBA. The MJACO is presented in section 3. In section 4, the ReMIOOP protocol is presented as a MIOOP extension for reliable multicast. Some implementation issues are described in section 5. In section 6, some experiments with our multicast ORB are presented. Finally, section 7 cites some related works, and in section 8 presents some final remarks of this research.

2 Group Communication in CORBA

Two significant initiatives were taken into account by OMG concerning the introduction of group communication mechanisms in CORBA. The first of them use the group abstraction to support fault-tolerant applications in object level (FT-CORBA [20,9]), and the other considers to use the ORB as a high performance group communication mechanism without reliability (UMIOP [19]). So, these two specifications can be considered complementary and indicate a trend in OMG, the attempt to specify a standardized group communication mechanism with differentiated guarantee levels for different applications.

The FT-CORBA standard, which introduced the concept of objects group in CORBA architecture, defines a set of object services that offer functionalities such as group management (membership), state transfer, fault detection and notification. One kind of support assumed by FT-CORBA, but not standardized by OMG, is the group communication service [25]. That specification defines that this service must support some communication properties in order to provide the underlying mechanisms for implementing active replication technique [27]; however, these specifications do not define the service semantics and protocols that must be implemented.

The UMIOP specifications, on the other hand, define an unreliable multicast service based on IP multicast. It can be considered as a basis for creating of a interoperable group communication mechanism standardized by OMG. Extensions for these specifications to define stronger properties for ordering and reliability would be appropriate for FT-CORBA standard.

2.1 UMIOP

In 1999, the OMG started a specification process for an unreliable multicast protocol based on IP multicast and objects group model to support this protocol in CORBA ORBs. This process culminated in UMIOP specifications release. This standard aims to support a multi-point communication mechanism in CORBA architecture, without any delivery guarantee. The protocol used by UMIOP is the MIOOP. This protocol maps GIOP messages into UDP/IP multicast. The basic function of MIOOP protocol is to segment and encapsulate GIOP messages, sent to the group, into packets. These packets contain a header (defined in the specifications) with a set of fields that allows the original message to be reassembled in the receiver side. Once the packets are properly arranged, the multicast

of message is made through UDP protocol, which provides an almost direct interface for IP services or in this case, for IP multicast. IP multicast defines a set of extensions to IP protocol enabling one-to-many communication (multicast). The main characteristics of this protocol are open groups (it is not necessary to be a member of the group to multicast a message to it), no membership (list of members), no reliability (such as IP) and accessibility through class D IP addresses (from 224.0.0.0 to 239.255.255.255).

The use of MIOP, and IP multicast in a subjacent level, make it possible to transmit the GIOP messages between two different ORBs. However, the conventional CORBA object model, which specifies that one object reference must correspond to only one object implementation, is not appropriate for object group. In despite of that, the semantics of CORBA point-to-point invocation is reliable concerning messages delivery, and the order is defined by sender, which can be configured with or without reply, unlike the MIOP definition. Therefore, a new object model representing groups had to be defined in UMIOP. This model does not define an object identifier, but a group identifier that can be associated with multiple object ids used by the POA (*Portable Object Adapter*) to activate the corresponding servants [19]. The semantics of messages delivery and ordering in UMIOP has no guarantees, and the MIOP supports only messages with no replies.

An objects group in UMIOP is composed by group identifier, and information about how to reach it in the communication network (class D IP address and a port). These information are contained into UMIOP group references detailed bellow.

2.2 UMIOP Group Reference

A reference or IOR (*Interoperable Object Reference*) is used to identify a single object in CORBA. Each IOR contains one or more profiles which allow the ORB to locate a servant object through any network transport mechanism. For example, IIOP profiles contain in its fields the server ORB address (usually, an IP address and a port) and an object identifier in the server ORB (object key), used to access implementations through TCP/IP.

In order to support groups, the UMIOP specifications define a group IOR that addresses a set of zero or more objects. A group IOR uses a different type of profile to send messages through UDP/IP multicast. This UIPMC profile, defined in the specification, contains all necessary information to access a multicast group (a class D IP address and a port) in transport level. Another structure, with the logical group identifier, is used for identifying members at ORB level. The group IOR can also hold two IIOP profiles: one for requests that demand reply and another that specifies a gateway to multicast requests when the client is unable to do that.

The figure 1 presents the complete group IOR format defined by OMG as part of UMIOP specifications. The composing of a group IOR must be made through the specification of information about the group and the IORs for group IIOP object and gateway. This creation is made with these information specification in a “corbaloc” URL, or through MGM (*Multicast Group Manager*) methods, an optional object service that provides operations for groups management.

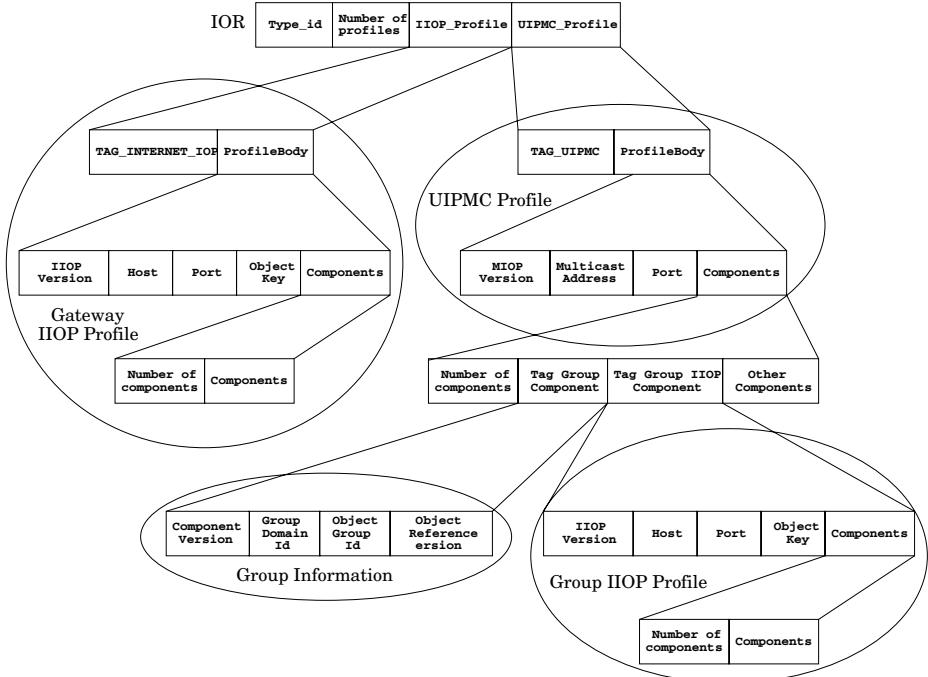


Fig. 1. Representation of the Group IOR

3 MJaco

From the UMIOP specification study we developed an ORB to fulfill these specification. This ORB was called MJACO [1,3], which is an extension of JACORB, a high-performance and open source CORBA ORB that implements CORBA 2.3 specifications (<http://www.jacorb.org>). The MJACO architecture was defined to allow the compatibility of two protocol stacks (IIOP/TCP/IP and MIOP/UDP/IP multicast) in the same ORB, contributing for better interoperability and portability.

Figure 2 illustrates the UMIOP and MJACO ORB integration architecture. In this figure the ORB is presented with the two protocol stacks: one for point-to-point communication based on IIOP using TCP/IP services, and other for multi-point communication based on MIOP, using UDP/IP multicast as transport mechanism. Our integration model presents some elements defined in the specification that compose the support for these two models of communication. Other components and extensions, not defined in that specification, has also been added. Their purpose is to facilitate the integration of the different stacks and to improve its efficiency.

The Multicast Adapter is a fundamental part of our integration model. It is responsible for managing the multicast sockets used in the reception of the MIOP packets and for the delivery of group messages to the POAs in the ORB.

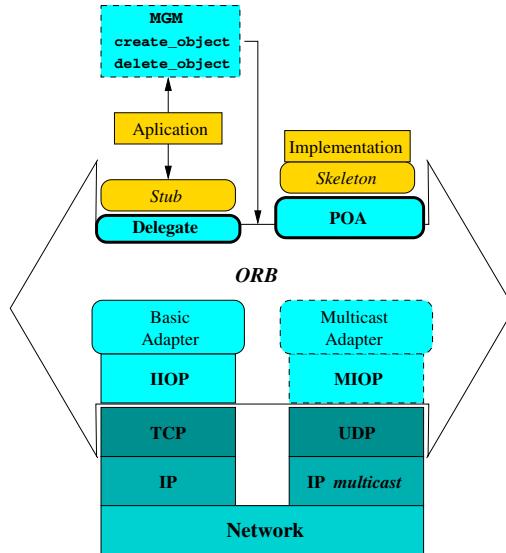


Fig. 2. MJACO Architecture

The POA and the Delegate are the main components of the ORB to be modified to add the UMIOP. The modifications on Delegate are made in some points to support multicast GIOP message to groups, since it is the first ORB internal component to be activated when a stub method call is executed. Four new methods for handling objects group, described in the OMG specifications, are added to the POA. In addition, the POA has to be modified in order to process requests addressed to groups. For every group upcoming request, a search in the active groups table is made in order to obtain the group member implementations to which the request is addressed in each POA.

4 ReMIOPIOP – Reliable MIOPIOP

As mentioned before, the MIOPIOP is unreliable, thus inadequate for many kinds of applications, like fault-tolerant or groupware systems, which do not allow message losses. We propose a set of extensions in the MIOPIOP, called ReMIOPIOP (*Reliable Multicast Inter-ORB Protocol*), which provides high probability in the message delivery guarantee. Basically, the ReMIOPIOP protocol is a receiver-initiated “best-effort” reliable multicast protocol (it uses NACKs to ask for lost message [14,24]) such as SRM [10], LRMP [15] and TRM [26]. The ReMIOPIOP may be seen as a minimal reliable multicast protocol in the sense that it uses only two very weak mechanisms - lost message recovery and flow control - in order to implement “best-effort” reliability.

The premise of ReMIOPIOP reliability properties assume a communication support with asynchronous systems characteristics, therefore, with no time limits guarantees for message transmission and remote operations execution. The fault

model considers only crash faults for hosts, and omission faults in the communication system. Another fundamental assumption concerning the consistency of the ReMIOP is the statement: after $Od + 1$ multicast of the same message, there is no correct host which did not receive this message (see section 4.1).

The ReMIOP operates as the following: messages (MIOP packets) are multicast by the sender to all receivers. The sender has no knowledge about the group members identity. The receivers detect lost packets by gaps in the sequence of received messages¹. When a member detects a missing packet, it multicast a control message (NACK) to the group, asking for the lost packets. Any member that receives this message, either the sender or any other receiver that owns the required packet, can multicast it to the group. This protocol also includes session messages that are multicast by the receivers to report to the senders its buffers state, allowing a dynamic adjustment of transmission rate through the flow control algorithm. The algorithm presented in the figure 3 describes the executed procedures to multicast and to receive messages through ReMIOP.

Before describing the algorithm of the figure 3 we shall explain some primitives used in it:

- *calculate_delay()*: This primitive is used to calculate the schedule time to the next multicast according to the flow control algorithm;
- *schedule_multicast(time, message)*: It is used to schedule the multicast of a message at a specified local clock time;
- *cancel_scheduled(message_id)*: Cancels the multicast of a message with specified id;
- *cancel_scheduled_nack_for(message_ids)*: Cancels the multicast of a NACK for specified messages. The *message_ids* is a set of message ids;
- *missing_messages(buffer)*: This function searches in the specified buffer for gaps in messages sequence and return a set with all the ids for these missing messages;
- *random(limit)*: This primitive chooses a random integer value between 0 and a specified limit;
- *nack(message_ids)*: Builds and returns a NACK requesting the messages with the specified ids;
- *update_send_rate(states)*: Applies the flow control rule to define the new send rate.

Besides these primitives, the algorithm for messages reception executes sequentially, and uses a scheduler that obeys the specified time schedule with minor deviations from this. The *R-multicast(m)* procedure, defined in figure 3, implements message transmission in two steps: the computation of the wait time to multicast the message (this calculation follows the flow control algorithm); and the unreliable multicast scheduling. The reception and delivery of messages is also illustrated in the figure 3. For reliable message delivery, we first receive it

¹ For the first message of each sender, the receiver creates a buffer for controlling sender messages.

```

procedure R-multicast(m):
   $T_d \leftarrow \text{calculate\_delay}()$  // Sender flow control
  schedule_multicast( $T_d, m$ ) // Message multicast scheduled

To R-deliver(m) do:
  U-receive(m)
  if m.type = DATA then // m.type: type of the message m
    cancel_scheduled(m.id) // Cancels m multicast, if scheduled
    if m  $\notin$  bufferm.sender then // m.sender: sender of m
      bufferm.sender  $\leftarrow$  buffer  $\cup$  {m}
      R-deliver(m)
      missing  $\leftarrow$  missing_messages(buffer)
      if missing  $\neq \emptyset$  then
        schedule_multicast(random( $T_{nack}$ ), nack(missing))
      end if
    end if
  else if m.type = NACK then
    cancel_scheduled_nack_for(m.nacked)
    for all buffers do // m.nacked: required messages list
      for all mr  $\in$  buffers : mr.id  $\in$  m.nacked
        if nacksmr  $\leq$  Od then
          schedule_multicast(random( $T_{repair}$ ), mr) // Repair
          nacksmr  $\leftarrow$  nacksmr + 1
        end if
      end for
    end for
  else if m.type = STATE then
    update_send_rate(m.states)
  end if

```

Fig. 3. Simplified ReMIOB algorithm

in an unreliable way using the *U-receive*(*m*) primitive. Only after that, the algorithm treats each of the three types of messages defined by ReMIOB protocol in a differentiated manner:

- If the incoming message is a data message (a GIOP message fragment) it is verified if this message was already received before ($m \notin \text{buffer}_{m.\text{sender}}$); if it is true, no action is taken. If the message was received for the first time, it is added in the reception buffer of this sender and then delivered to the application. After that, receivers verify if there are missing messages and multicast NACKs on the group for error recovering;
- If the received message is a NACK, then the NACK suppression mechanism is activated (cancelling NACKs that was already received). For each requested message id in the NACK a repair message is prepared (when the receiver is capable to repair it). The retransmission of this message is scheduled

- for a posterior time randomly defined, which avoids an explosion of repair messages. Note that a repair message multicast is conditioned to the Od limitations (see next subsection);
- Finally, if the message is a state notification from a group receiver (buffers state), the system takes this state into account to update the transmission rate of the protocol.

The mechanisms used to add reliability to ReMIOP are detailed in the next subsections.

A Note on Garbage Collection: As already mentioned, the ReMIOP implements probabilistic reliability mainly because, in real systems, it can not maintains the received messages into the buffer indefinitely. The buffers used in the algorithm of figure 3 (represented as a set) are not infinite. They have a predefined fixed size and when this limit is reached, the older messages are thrown away. The subsection 4.2 discuss some more issues related to the buffers of the ReMIOP.

4.1 Lost Message Recoveries

As the possibility of message loss exists and may be substantial, specially in large scale systems, the ReMIOP includes a kind of control message to allow requests for retransmitting lost messages: NACK messages. This message contains the identifiers of lost MIOP packets, so that host, which receives the message, is able to multicast the asked packets. Despite this mechanism, characterized for being initiated by receiver, some improvements had been added to the protocol in order to prevent, as possible, flooding of NACKs and repair messages. Among these modifications it can be mentioned the use of a RINA (*Receiver Initiated Nack Avoidance*) mechanism [24] and repair delay. These two improvements cause a delay on the diffusion of NACKs and repairs for random periods of time in the expectation that another group member does the retransmission.

Moreover, the omission degree parameter (Od) was introduced as an *optional improvement*. In this case, the sender and the receivers involved in the group interactions can retransmit the same message again until the limit defined by $Od+1$ is reached. In communication supports with omission faults it is acceptable to consider that no more than Od retransmissions of one single packet is lost in a reference period of time. Tests can be executed in real networks to determine Od in any degree of probability [28]. If a receiver does not receive a packet after $Od+1$ transmissions from a sender, then it is possible to assume that the receiver is faulty (*crash*). Note that Od is only one parameter that can be used in the protocol, even when the system assumed to be asynchronous. If Od is a too high value, then the protocol will execute like those ones which assume *reliable channels* [12]. For this environment (since it is asynchronous, bursts of messages may be over-delayed, instead of lost) this artificial hypothesis (omission degree) can make a too slow process (or slowly connected) be treated as a crashed one. This hypothesis can be considered acceptable because it allows progress of the protocol, however this method is subject to inconsistencies if failures are not

correctly detected. Therefore, this parameter is useful only for practical ends. As mentioned earlier, the omission degree is an optional improvement, so, if it is not used, then, in our algorithm, the Od variable is set to ∞ .

4.2 Flow Control

Flow control is a fundamental mechanism for any reliable multicast service. The absence of membership information in ReMIOP environment makes impossible the use of more refined flow control algorithms like those defined in [5]. Therefore, we use a simple mechanism, inspired in LRMP flow control [15], that provides packets loss prevention in hosts, and the consequent NACKs explosion.

The mechanism applied to implement the flow control in the ReMIOP protocol uses information provided by NACKs (that contain lost messages indication) and state messages (that contain the reception buffers state of the members) received². Through these information the sender can estimate the speed of its receivers and can apply a rate update function according to the receivers capacity. This mechanism uses two types of buffers: one for senders and another for receivers.

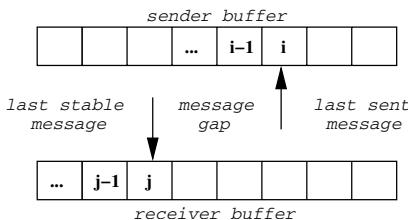


Fig. 4. ReMIOP flow control

In figure 4, the sender buffer is used to store sent messages as well as messages to be sent. The size of this buffer determines how many old messages can be resent by this host in case of a NACK reception. In the receiver, the buffer stores the received messages. The difference between the sequence number of the last message sent by the sender (i in sender buffer) and the last received stable message (j in receiver buffer) is the parameter used to adjust the sending rate.

Let the difference $\delta = i - j$ be such that: the bigger is δ , the lower must be the sending rate so that slow receivers (whose j is much lesser than i) are able to consume the sender messages. The main objective of this algorithm is to provide transmissions adjusting the sending rate in a manner that all group members, including the ones that are in congested areas of the network, can receive the messages.

The transmission rate of the senders always varies between the interval $[R_{min}, R_{max}]$. Where R_{min} and R_{max} are defined by the application. The initial rate is defined as $R_0 = (R_{min} + R_{max})/2$. We consider the adjustment

² Each receiver has a reception buffer for each sender.

band as $R_{max} - R_{min} = L$ in which, with a discretization we assume ten levels of transmission rate: $R_{max} - R_{min} = 10(0.1L)$. For each $size/10$ packets sent, the sender adjusts the transmission rate R according to the following rule (where $size$ is the fixed size of the sender buffer, i.e. its maximum capacity):

$$R_i = \begin{cases} R_{min} & \text{if } \delta > \frac{size}{2} \\ R_{i-1} + (-1)^{\lceil \frac{\delta - \frac{size}{5}}{size} \rceil} 0.1L & \text{otherwise} \end{cases} \quad (1)$$

In the equation 1, R_i defines a new sending rate based: (i) on the current rate (R_{i-1}), (ii) on the greater δ collected in the period and (iii) on the size of the sender buffer ($size$). This adjustment rule states that the sending rate will always be determined by slowest receiver.

Through this flow control algorithm and the (informal) assumption that the system operates most of the time in normal conditions (without congestions and omission faults), it is possible to guarantee that all the messages sent will be delivered to the group members.

5 ReMIOP Implementation

In order to implement ReMIOP as a communication service of MJACO, we have to consider three important issues: control messages definition, ReMIOP/MIOP interoperability and where in the protocol stack we will implement the ReMIOP algorithm (presented in figure 3). This section considers these and others issues.

5.1 Control Messages Definition

As mentioned before, the ReMIOP reliability is supported by two important mechanisms: message recovery and flow control. Each of these mechanisms requires some type control messages, that are defined in the figure 5.

In this IDL two types of messages are defined: NACK and STATE. These two kinds of messages are defined by the `MessageType` enumeration. The `ReMIOPControl` structure defines the fields of the control messages used by protocol. The first field of this structure defines the type of message, following this, the `senderId` field contains the IP address of the control message sender. The last field of the structure, the `messages` field, have different purposes depending the type of the message: if it is NACK, then this field contains an array of ids of messages (represented by the `MessageId` structure) identifying the required messages that this host has lost. Otherwise, if it is a session message, then this field contains the maximum stable message for each sender of the group.

Independently of what type of the control message defined in the `ReMIOPControl` structure, it will be serialized such as any other IDL definition and encapsulated in MIOP packets as stated in the next subsection.

```

module ReMIOP {
    enum MessageType {
        NACK, STATE;
    };
    struct MessageId {
        string senderId;
        unsigned long long sequenceNumber;
    };
    typedef sequence<MessageIds> MessageIds;
    struct ReMIOPControl {
        MessageType type;
        string senderId;
        MessageIds messages;
    };
}

```

Fig. 5. Extension on UMIOP: ReMIOP Messages

5.2 ReMIOP/MIOP Integration

To ensure the interoperability requirement, the data packets sent by ReMIOP are exactly the same as the ones sent by MIOP. So the data sent by ReMIOP can be received by MIOP receivers as well.

Each ReMIOP control message is encapsulated in one MIOP packet and transmitted to the IP multicast group just like a data packet. However, MIOP receivers do not process these control packets, they are ignored. That means that basic issues must be considered when fulfilling the MIOP packets header fields to ensure they are discarded.

The filling of MIOP packets header containing ReMIOP control messages obeys some basic rules:

1. the packet id must have the same value, and this value cannot be used by data messages;
2. the packet number field of the header is always set to 0 value;
3. the field defining the number of packets that are part of this message is always set to 2 value;
4. a bit is marked in the field `flags` indicating that the packet is a ReMIOP control message.

The rest of the fields of the MIOP packet are filled in conventional way according to the MIOP protocol specification.

If the MIOP header fields are set as indicated above, the receivers of ReMIOP packets capable to process it will detect flag pointed in the `flags` field and will treat them adequately (as a ReMIOP control packet). The receivers that do not implement ReMIOP will process the packet as the first element of a size 2 collection of packets. As the second packet of this collection do never arrives, it is never released to the ORB upper layers, and will be discarded after a timeout, just as defined by UMIOP specifications [19].

5.3 Plugable Strategies for MJaco

In order to make available a reliable multicast service provided by ReMIOP in MJACO, a plugin mechanism was implemented. This mechanism allows the integration of reliability strategies capable to extend the ORB multicast stack. So, many other protocols, with distinct features, could be implemented over this unreliable multicast service. The figure 6 illustrates the architecture of this mechanism.

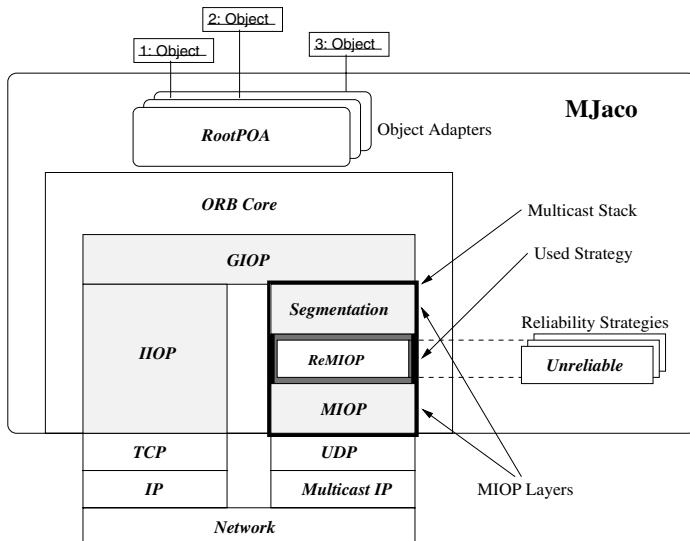


Fig. 6. MJACO architecture with plugable strategies

In figure 6, we have the reliability strategy loaded as a multicast stack layer. This layer, which can be plugged, is in between two other layers (segmentation and MIOP) which characterizes the MIOP implementation. In a lower level layer we have the MIOP encapsulating data blocks in MIOP packets and transmitting them using UDP/IP multicast. The layer above the plugin is a segmentation layer, responsible for disassembling (marshalling) long messages in collections of MIOP packets and reassembling (unmarshalling) them on the receivers.

6 Results

In order to verify the performance of our reliable multicast service in a CORBA ORB, we executed a set of simple comparative tests concerning the use of MJACO with the ReMIOP strategy, pure MIOP, and using multicast sockets. These tests had been accomplished in four equally configured machines³ in a LAN with

³ Pentium IV 1.6GHz, 256 Mbytes of RAM memory, Mandrake Linux 9 Operating System (kernel 2,4) and 100Mbps Ethernet network card.

minimal external network load. The test objective was to validate our implementation measuring the MJACO+ReMIOp performance, so more complex network architectures were not considered. The test program measured the time needed by a group member to multicast a variable size message and receive the reply message from all group members (including itself). It is called round trip time⁴.

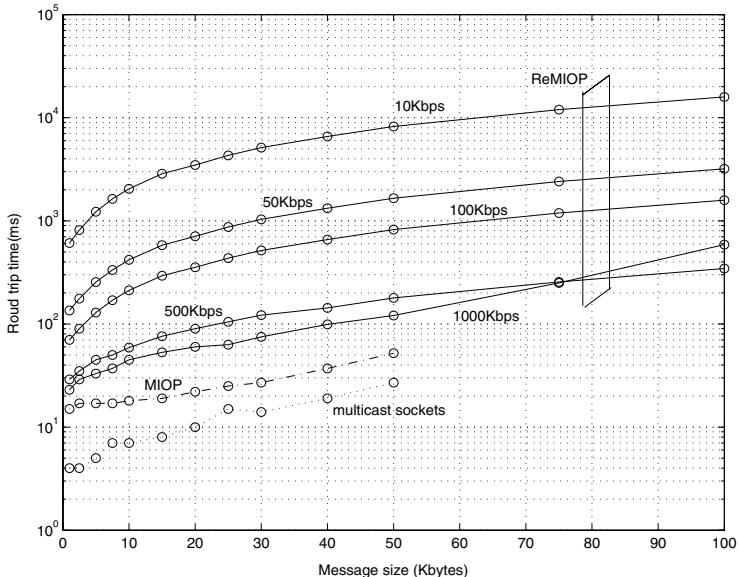


Fig. 7. MJACO performance

Several experiments were executed with different values for R_{max} (see section 4.2). R_{max} defines the maximum transmission rate (in bits per second) of the ORB to multicast messages to a group. The larger R_{max} is, the faster is the transmission, and greater is the possibility of message losses. Experiments with MJACO using pure MIOp and with multicast sockets were also run to find out the costs of reliable multicast in middleware level. The graph of figure 7 presents these experiments results in a logarithmic scale.

Figure 7 shows that the higher the value of R_{max} the closer is the behavior of ReMIOp to the original MIOp. The behavior described by the curves with lower R_{max} values (10Kbps, 50Kbps and 100Kbps) is extremely reliable, so, in our experiments, no packet was lost, and therefore, no NACK was multicast to the group. However, the round-trip times obtained are much larger than when using MIOp since each sender waits much longer to transmit each packet. In fact, the perceived round-trip overhead is a direct consequence of the value of R_{max} .

⁴ All communication (message and replies) are done via the tested multicast protocol: ReMIOp, MIOp and UDP/IP multicast.

and the number of data packets⁵ (message size). The ReMIOP layer overhead (compared with MIOP) is about 5ms per round-trip for small messages (one data packet) and large values of R_{max} , as can be seen in figure 7.

In curves with bigger values of R_{max} , the round trip time is lower, and the amount of lost packets, NACKs and transmissions is higher. In the curve with $R_{max} = 1000Kbps$, the round trip time is equal or higher than that of the curve $R_{max} = 500Kbps$ for long messages ($\geq 75K$). This result is caused by the great amount of lost packets (NACKs also) and repairs.

The graph of figure 7 also describes curves with the behavior of the MIOP and multicast sockets to round trip time. These curves describes messages of at most 50 Kbytes, because for longer messages the packet losses disable the use of our test program since the round-trip cannot be completed.

7 Related Work

There are some works aiming to incorporate group communication in CORBA architecture [17,8,18]. These works, in general, focuses in integrating existing group communication systems and protocols in CORBA middleware without concerning interoperability issues⁶. However, only recently, with the publication of UMIOP specifications, the possibility of implementing interoperable group communication mechanisms based on IP multicast had been opened. A similar study of ours is the RMIOP protocol [23] that proposes an extension of MIOP for reliability purposes. This protocol uses a NACK-based approach and ACKs messages to signal the acknowledgement of each received GIOP message (each receiver sends an ACK to the sender after receiving all the packets from a collection). This type of policy needs membership information, so the sender has to know who are the members of the group in order to collect the ACKs. The use of membership protocols increases the complexity of the RMIOP and degrades the performance in highly dynamic groups (where processes join and leave groups all the time). The approach presented in this work does not need membership, and in our conception, this type of service must be implemented at object level, using the FT-CORBA facilities (like `ObjectGroupManager` interface) [20]. Moreover, in [23] mechanisms of flow control are not used (or not reported). The RMIOP implementation was made in C++ on ORBacus 3.1, a proprietary ORB that provides sufficiently generic plugin mechanism for integrating new transport protocols. We used an open-source ORB because of the possibility of unrestricted extensions implementation.

In [11] it is described another approach integrating reliable multicast in a CORBA ORB that was developed by the same group that idealized the RMIOP. In that work the reliable multicast is provided through a LRMP library integrated to the ORBacus as a plugin. Therefore, it is not a standardized and

⁵ According to the flow control algorithm, some packets must wait some time before being sent.

⁶ An excellent review of some of these efforts for fault-tolerance is the paper by Felber and Narasimhan [9].

interoperable solution. It is important to remember that although it represents a simple solution, this work precedes the publication of UMIOP specifications and it brought some ideas that were included later in that specification.

The literature for “best effort” reliable multicast protocols is broad [14]. These protocols are characterized by the absence of acknowledgements message and scalability. As some of the protocols of this type we can mention the SRM [10] and TRM [26] and LRMP [15], this last one has great influence on the ReMIOP conception. Despite the mentioned successful protocols, the IETF (Internet Engineering Task Force) did not adopt any of them as the standard multi-point reliable transport protocol for the Internet. Instead, they defined a series of mechanisms that compose a reliable protocol so that different applications, with distinct requirements, can build a variety of protocols from standardized mechanisms [13,29].

Two OMG specifications are related with our work in different aspects. The already mentioned ROMIOP (Reliable and Ordered Multicast Inter-ORB Protocol) upcoming specifications [21] may have an important role as it evolves, since it defines formats for various types of control messages used in multicast protocols (e.g. ACKs, NACKs and order enforcement). Another OMG specification of great interest is the ETF (Extensible Transport Framework) [22]. These specifications define a framework to integrate new transport protocols in CORBA ORBs. Unfortunately this framework assumes that the protocol to be integrated is reliable, point-to-point and connection-oriented. These assumptions make it difficult to implement group communication protocols as ETF plugins. The work presented in [4] is an initial attempt to integrate these two specifications devising an interoperable total order multicast protocol. This kind of protocol satisfies more restrictive properties and consequently is much heavier than ReMIOP.

8 Conclusions

The main objective of this paper was to propose extensions to standardized MIOP specification to obtain a reliable multicast support. The resulting protocol is better suited to implement more restrictive ordering and agreement guarantees and is developed as part of a CORBA ORB. The integration model, which uses the plugins mechanisms, is very flexible and do not compromise the ORB interoperability and portability aspects. The ORB is capable to make invocations using ReMIOP, MIOP or IOP.

The ReMIOP implementation makes possible the development of other CORBA architecture group communication solutions (such as [2,4]). These solutions are being used in GROUPPAC project [16,2], which implements the Fault-Tolerant CORBA specification.

Despite of that, it was also presented some ReMIOP implementation performance measures, comparing it to MIOP and IP sockets multicast to verify the costs related to this service quality available on middleware level. These developments, which were based on the proposed integration model, were built on JACORB. These implementations can be obtained on the web in the following address: <http://grouppac.sourceforge.net/>.

References

1. Alysson Neves Bessani, Joni da Silva Fraga, and Lau Cheuk Lung. Implementing the multicast inter-ORB protocol. In *Proceedings of the 6th IEEE International Symposium on Object-oriented Real-time distributed Computing - ISORC'03*, Hakodate - Hokkaido - Japan, 2003.
2. Alysson Neves Bessani, Joni da Silva Fraga, Lau Cheuk Lung, and Eduardo Adilio Alchieri. Active replication in CORBA: Standards, protocols and implementation framework. In *Proceedings of International Symposium on Distributed Objects and Applications (DOA'04)*, volume 3291 of *Lecture Notes in Computer Science*, Larnaca, Cyprus, October 2004. Springer-Verlag.
3. Alysson Neves Bessani, Lau Cheuk Lung, and Joni da Silva Fraga. Integrating the unreliable multicast inter-ORB protocol in MJaco. In *Proceedings of the 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems - IFIP DAIS'03*, volume 2893 of *Lecture Notes in Computer Science*, pages 200–211, Paris - France, 2003. Springer-Verlag.
4. Daniel Borusch, Lau Cheuk Lung, Alysson Neves Bessani, and Joni da Silva Fraga. Integrating the ROMIOP and ETF specifications for atomic multicast in CORBA. In *Proceedings of International Symposium on Distributed Objects and Applications (DOA'05)*, Lecture Notes in Computer Science, Larnaca, Cyprus, October 2005. Springer-Verlag.
5. Raimundo José de Araujo Macêdo, Paul D. Ezhilchelvan, and Santosh K. Shrivastava. Flow control schemes for a fault-tolerant multicast protocol. In *Proceedings of Pacific Rim International Symposium on Fault-Tolerant Systems (PRFTS'95)*, Newport Beach, California, USA, December 1995. IEEE Computer Society.
6. S. E. Deering. Host extensions for IP multicasting (rfc 988). IETF Request For Comments, July 1986.
7. S. E. Deering and D. R. Cheriton. Host groups: A multicast extension to the internet protocol (rfc 966). IETF Request For Comments, December 1985.
8. Pascal Felber, Benoît Garbinato, and Rachid Guerraoui. The design of a CORBA group communication service. In *Proceedings of the 15th Symposium on Reliable Distributed Systems (SRDS'96)*, pages 150–159, Niagara-on-the-Lake, Canada, 1996.
9. Pascal Felber and Priya Narasimhan. Experiences, strategies, and challenges in building fault-tolerant CORBA systems. *IEEE Transactions on Computers*, 53(5):497–511, 2004.
10. S. Floyd, V. Jacobson, C.-Gung Liu, S. McCane, and L. Zhang. A reliable multi-cast framework for light-weight session and application level framing. *IEEE/ACM Transactions on Networking*, December 1997.
11. C. Gransart and J.-M. Geib. Using an ORB with multicast IP. In *Proceedings of PCS99: Parallel Computing Systems Conference*, Ensenada - Mexico, 1999.
12. V. Hadzilacos and S. Toueg. A modular approach to the specification and implementation of fault-tolerant broadcasts. Technical report, Department of Computer Science, Cornell University, New York - USA, May 1994.
13. M. Handley, S. Floyd, B. Whetten, R. Kermode, L. Vicisano, and M. Luby. The reliable multicast design for bulk data transfer (rfc 2887). IETF Request For Comments, August 2000.
14. B. N. Levine and J. J. G.-L.-Aceves. A comparison of reliable multicast protocols. *Multimedia Systems*, 6(5):334–348, 1998.

15. T. Liao. Light-weight reliable multicast protocol. Available at <http://webcanal.inria.fr/lrmp/>, 1998.
16. Lau Cheuk Lung, Joni da Silva Fraga, Jean-Marie Farines, Michael Ogg, and Aletta Ricciardi. CosNamingFT - a fault-tolerant CORBA naming service. In *Proceeding of the 18th International Symposium on Reliable Distributed Systems - SRDS'99*, Lausanne - Suice, 1999.
17. Silvano Maffeis. Constructing reliable distributed communication systems with CORBA. *IEEE Communications Magazine*, 14(2), 1997.
18. L.E. Moser, P.M. Melliar-Smith, P. Narasimhan, R.R. Koch, and K. Berke. Multicast group communication for CORBA. In *Proceedings of International Symposium on Distributed Objects and Applications*, pages 98–107, Edinburgh, United Kingdom, September 1999.
19. Object Management Group. Unreliable multicast inter-ORB protocol specification v1.0. OMG Standart ptc/03-01-11, October 2001.
20. Object Management Group. The common object request broker architecture: Core specification v3.0. OMG Standart formal/02-12-06, December 2002.
21. Object Management Group. Reliable, ordered, multicast inter-ORB protocol (revised submission). OMG TC Document realtime/2003-10-04, October 2003.
22. Object Management Group. Extensible transport framework v1.0. OMG TC Document ptc/2004-01-04, January 2004.
23. S. L. Dit Picard, S. Degrande, and C. Gransart. A CORBA based platform as communication support for synchronous collaborative virtual environments. In *9th ACM Multimedia Conference*, Ottawa - Canada, 2001.
24. S. Pingali, D. Towsley, and J. F. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *Proceedings of the Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 221–230, New York, NY, USA, 1994. ACM Press.
25. David Powel. Group communication. *Communications of the ACM*, 39(4):50–53, April 1996.
26. B. Sabata, M. Brown, B. Denny, and C. H. Heo. Transport protocol for reliable multicast: TRM. In *Proceedings of the International Conference on Networks*, Orlando - Flórida - USA, 1996.
27. F. B. Schneider. Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.
28. P. Veríssimo, L. Rodrigues, and A. Casimiro. Cesiumspray: a precise and accurate global clock service for large-scale systems. *Journal of Real-Time Systems*, 12(3):243–294, 1997.
29. B. Whetten, L. Vicisano, R. Kermode, M. Handley, S. Floyd, and M. Luby. Reliable multicast transport building blocks for one-to-many bulk-data transfer (rfc 3048). IETF Request For Comments, January 2001.

Integrating the ROMIOP and ETF Specifications for Atomic Multicast in CORBA*

Daniel Borusch¹, Lau Cheuk Lung¹, Alysson Neves Bessani²,
and Joni da Silva Fraga²

¹ Graduate Program in Applied Computer Science,
Pontifical Catholic University of Paraná,
Curitiba - PR - Brazil

{dborusch, lau}@ppgia.pucpr.br

² DAS – Departamento de Automação e Sistemas,
UFSC – Universidade Federal de Santa Catarina,
Florianópolis - SC - Brazil
{neves, fraga}@das.ufsc.br

Abstract. OMG published a draft specification for a reliable ordered multicast inter-ORB protocol to be used by distributed applications developed in CORBA (ROMIOP). This specification was made to attend the demand of applications that needed more restrictive guarantees on reliability and ordering, since there already has a specification without these resources (UMIOP). This paper presents how ROMIOP was implemented, as well as modifications that were made on the specification to make possible to implement it according to the ETF (Extensible Transport Framework) specification. Performance measures were made comparing ROMIOP with others protocols, like UMIOP, to show its characteristics and its cost.

1 Introduction

The CORBA (Common Object Request Broker Architecture) [15] architecture, standardized by OMG (Object Management Group), has the ORB (Object Request Broker) as its main component. It makes possible that objects receive and make invocations in a transparent way in distributed systems, being considered the base for the interoperability between applications on heterogeneous environments. To accomplish the exchange of messages between ORBs, there is an element that specifies a default transfer syntax besides a set of messages formats known as GIOP (General Inter-ORB Protocol). The implementation of GIOP to the TCP/IP protocol is known as IIOP (Internet Inter-ORB Protocol), which uses point-to-point communication as base, ideal for client/server applications. However, several different application areas need to disseminate the same message to an infinity of hosts. One of the ways to do this is using multicast IP, which contains a set of extensions to the IP protocol that make possible to realize multipoint communications [4].

Since there was not a specification that described a way to use multipoint communication in CORBA architecture, in 2001 OMG published the UMIOP

* This work is supported by CNPq (Brazilian National Research Council) and FA (Fundação Araucária) through processes 481523/2004-9, 506639/2004-5, 401802/2003-5 and FA-6651/04.

(Unreliable Multicast Inter-ORB Protocol) [18, 1] specification. UMIOP was proposed to provide a common mechanism to deliver requisitions by multicast, without offering delivery guarantees (reliable multicast) and even less total ordering. The standard transport protocol defined for UMIOP was multicast IP over UDP/IP, that differently than TCP/IP, is not connection guided. With UMIOP only one-way (without answer) invocations can be accomplished. Not having any kind of guarantee, UMIOP can be characterized as being of high performance, making it ideal for applications like audio and video streaming, where the loss of some packets can be tolerated. However, several applications cannot tolerate packet losses, needing more restrictive guarantees like agreement and ordering. Because of that, and since OMG had not published until that moment a specification that came with a solution, our research group proposed the ReMIOP (Reliable Multicast Inter-ORB Protocol) [3] protocol to supply that demand.

However, at the end of 2002, OMG published a draft specification, planned to be standardized on 2005. The specification introduced a solution using a multipoint communication protocol with delivery guarantee and total order (in other words, atomic multicast [5]). This specification was named ROMIOP (Reliable, Ordered, Multicast Inter-ORB Protocol) [17, 14]. ROMIOP, just like UMIOP, also uses multicast IP over UDP/IP, however invocations with return of answer (two-way) are supported. Making a detailed study of ROMIOP it is possible to verify that the specification only provides a series of IDL interfaces some of them still confused, giving a large space for interpretations. In other words, the specification does not supply details about how the interfaces must be implemented or which ordering algorithm should be used [5].

Besides this specifications, OMG recently published the ETF (Extensible Transport Framework) [16] specification, which defines a framework that allows anyone to project and implement additional transport protocol plug-ins of GIOP messages on ORB. This specification, which is already implemented in the majority of available ORBs, makes possible the extension of an ORB/CORBA with the addition of new transport protocols without having to make any significant modification in its structure. However, ETF was conceived aiming only point-to-point transport protocols. For multipoint transport, extensions in the specification are necessary.

This paper proposes, as its main contribution, a set of extensions to effectively integrate the ROMIOP and ETF specifications, including architectural and conceptual aspects besides some project decisions. The proposed solution is completely interoperable, without the use of proprietary interfaces, and totally in accordance with the OMG specifications. With this, we can consider it close enough to what could be a definite solution in terms of group communication with total order in CORBA. We also defined an atomic multicast algorithm to be implemented inside ROMIOP as a way to validate the proposed architecture, and finally we did some measures showing the cost of total ordering inside ORB.

This paper is organized in the following way: on section 2 some related works are shown. On section 3, the MJaco architecture and a set of extensions to the ROMIOP and ETF specifications are presented. On section 4, the used algorithm of total order is introduced. The section 5 presents some considerations regarding the implementation. Some results obtained with ROMIOP can be seen on section 6. Finally, section 7 presents the conclusions of this work.

2 Related Works

Group communication in CORBA was and still is a very interesting subject. The first works regarding this issue used proprietary tools of group communication. These works can be classified in the literature into three basic solutions: the approach on integration [10], on service [6, 7] and on interception [11, 12].

The integration approach consists in the construction or modification of an existent ORB, adding ways to make group processing. The main idea in this approach is that the group processing should be supported by a group communication underneath the ORB core. On the other hand, the approach that uses service objects is to provide the support for objects groups as a set of services on top of the ORB, and not as a part of the ORB. Finally, the interception approach forecasts that messages sent to the servers' objects must be captured and mapped into a group communication system, in a transparent way to the application.

In [2] it is proposed an implementation of atomic multicast over MJaco. This implementation uses the MIOP and IIOP protocols in the development of a state machine replication system [19] optimal in several aspects. A negative point about this work, in comparison with ROMIOP, is that it depends on the FT-CORBA infrastructure [13, 8], implemented through the GroupPac system.

This proposal has the virtue of having available the last OMG specifications related with multicast in CORBA. The use of these allowed us to achieve a complete interoperability and portability on ORB, fundamental requirements of any OMG specification.

3 MJaco Architecture

MJaco [1] is a CORBA middleware with group communication support based on the UMIOP [18] specifications, standardized by OMG. This middleware allows the multicast of messages in a non-reliable way, in accordance with the UMIOP standard, or reliable, implemented by the ReMIOP [3] and ROMIOP protocols, all three being based on the UDP/multicast IP stack. The integration model allows all protocols to be added to the ORB without changing the properties of portability and interoperability.

In figure 1, we have the ORB with two protocol stacks: one for point-to-point communication, based on IIOP, utilizing the TCP/IP services, and the other for multipoint communication, made by MIOP, ReMIOP and ROMIOP, utilizing UDP/multicast IP. The integration model presents several elements defined in the specification that composes the support for the two communication models.

The first stage of the MJaco project was the integration and implementation of UMIOP in the ORB. The next step on this project was the implementation of the ReMIOP protocol, which extends the UMIOP specifications with the property of reliable multicast, providing a "best effort" guarantee that all sent messages will be delivered by all correct processes of a group. Finally, ROMIOP was added to the set of protocols, being the only one that, besides having reliable multicast, provides total order message delivery, in other words, all correct members deliver all messages in the same order.

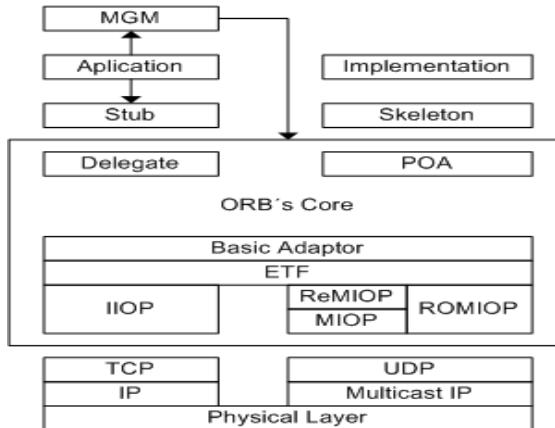


Fig. 1. MJaco architecture

It is important to note the way ROMIOP was introduced in MJaco. Instead of putting it on top of ReMIOP, or even in the same level as ReMIOP using MIOP as base, it was developed from the bottom. This, as will be shown later on, was made due to considerable differences on the format of the specifications of these protocols, being preferable to start its implementation without using almost anything from MIOP. It is also important to clarify that the ETF specification does not easily allows stacking protocols.

3.1 ETF Specification

ETF [16] is a specification of a platform that enables third parties to project and implement messages transports plug-ins. With it, several middleware that implement CORBA become much more flexible. This happens because there is in the specification all interfaces and methods that must (or may, on the optional ones) be implemented and which functionality each one must have, making unnecessary to modify the ORB code. This ensures the proprieties of portability and interoperability of the ORB.

The biggest problem of this specification is that it only defines how to add point-to-point transport plug-ins, which makes it deficient to multipoint protocols, like ROMIOP. Due to this fact, the chosen middleware had to be slightly modified, adding the functionality of sending messages to groups.

Basically, the specification defines four mandatory interfaces: connection, profile, listener and factories. The first one splits the message layer (GIOP) of the ETF layer, creating an interaction channel between messages and connections (both from clients and servers). The second one stores all the information related to the protocol, including methods to send (marshalling) and receive information through the IOR. The third one provides the initiative to “be connected” to a requisition made by a client, directing this requisition to a server. The forth and last interface is responsible to create instances of clients, making the connection of the ORB with the plug-in (the figure 5, on section 5, presents the ROMIOP implementation as an ETF plug-in).

A sequence of steps showing the interaction of a plug-in and an ORB can be seen in the figure 2 bellow:

1.	(Creating Server Listener)		
2.	ORB	->	Factories.create_listener()
3.	ORB	->	Listener.set_handle()
4.	ORB	->	Listener.listen()
5.	ORB	->	Profile.marshal()
6.	(Creating Client Connection)		
7.	ORB	->	Factories.demarshal()
8.	ORB	->	Profile.is_match()
9.	ORB	->	Factories.create_connection()
10.	ORB	->	Connection.connect()
11.	(Creating Server Connection)		
12.	Handle.add_input()	<-	Plugin
13.	ORB	->	Listener.accept()
14.	(Request Messages Receipt at Server)		
15.	Handle.data_available()	<-	Plugin
16.	ORB	->	Connection.read()

Fig. 2. Interaction steps between the ORB with the plug-in

At first there must be an instance of the server running that stays waiting the creation of client connections (line 1). All these steps are made by the ORB, starting with the invocation of the method `create_listener` provided by the factories interface (line 2). It creates an object that implements the `listener` interface and it is returned to the ORB. In the next two lines (3 and 4) the ORB uses the return of line 2 to invoke the methods `set_handle` and `listen`. The first one simply allows the plug-in to callback the ORB whenever it is necessary while the last effectively allows this instance to receive requests. Lines 5 and 16 indicate that at any time the profile of this transport can be marshaled (be serialized).

The next stage to make a communication is the creation of a connection requested by the client (line 6). All these steps are made by the ORB, starting with the invocation of the method `unmarshal` provided by the object that implements the factories interface (line 7). This happens just after the protocol being used is identified by the information contained in the received IOR. This invocation returns an object that implements the `profile` interface, that is used by the next line of code (line 8) to verify if it does not already exists an equal profile created before, meaning that the connection has already been opened before. If the connection does not exist, then both following instructions (lines 9 and 10) will be called. The first one, provided by factories interface, creates a connection, while the second uses the return of the first one to effectively enable the connection.

The third stage is the creation of the server side of the connection, which always happens when a client requests a new connection (line 11). There are two distinct possibilities to accomplish this function: by “callback” and by “polling”. The first

option (line 12) is initiated by the plug-in that calls the method `add_input` of the `handle` interface that was received as parameter on line 3. On the other hand, the second option (line 13) is initiated by the ORB invoking the method `accept`, which stops the thread until there is a connection.

The last stage shows the receipt of a request message by the server side (line 14). After the establishment of the connection in the last stage, the `listener` instance can signal to the ORB that there are new data available, calling the method `data_available` of the `handle` interface (line 15). Next, the ORB can read these data through a `read` method (line 16) of the `connection` interface.

3.2 ROMIOP Specification

ROMIOP (actually a draft [17, 14]) defines a set of interfaces to provide a multi-point communication with delivery guarantee and total order for every non-faulty members of a group of objects. It supports both requests that need replies (two-way requests/replies) and the ones that do not need (one-way requests). This specification was projected so that the protocols that implement it could coexist with IIOP, MIOP and any other multicast communication protocol, not being able to interfere in the functioning of them.

The specification defines an interface that configures the several available methods to consolidate replies and the quality of the ordering service. Regarding the first factor, there are two distinct ways to accomplish the consolidation: **simple voting**, where there are three possibilities to determine the reply (first received, last received and the first that satisfy the parameter of data consistency); **quorum voting**, where there are two possibilities to determine the reply (number of members that send the same reply and percentage of members that send the same reply), both dependent of the consistency parameter.

The data consistency parameter settles three possibilities: all the replies must be the same; all the replies must be different (apparently without any utility); and the standard, in which the majority of the same replies prevail. It is interesting to note that with this last option it is possible to provide a limited support to fault tolerance, using the idea of state machine replication [19]. Beside that, there is an additional parameter related to the reply consolidation that ends up overcoming all the others. It is possible to configure a timeout to the receipt of the replies. With this enabled, even if the chosen option has not yet been accomplished, the consolidation process is forced with the already obtained results.

The specification also defines how the consolidation and notification of replies must work: to each request message sent to the objects group there must be created an instance of a class responsible for the receipt of its replies. This instance is responsible for determining if the reply is a success, if a timeout happened, if there is no sufficient quorum, if the voting was inconsistent or if a key member was missing.

Another foreseen interface in the specification is the group service that provides basic operations like add and removal of members, besides the creation of groups.

To finish, speaking about the communication protocol, ROMIOP defines only formats for several types of messages. These formats consist in a header that has a fixed size and brings information related with the data contained in the packet (like its type, number of identification, position of the packet inside a set of messages, etc.)

and an area of data. The defined types of packets are Request, Reply, ACK, NAK, KeepAlive, Cancel and MemberChange. Each type has a different structure in the data area, following the semantic of the packet. Between several fields we can quote: to which member the packet is directed, address of ACK/NAK, ID of the already received packet, status of a member, etc.

Each one of the packets described above always come after a standard packet header (structure `PacketHeader`) that inform, besides the type of the following packet, complementary information, like if there is the need to send an ACK after the receipt of the same, the size of the packet unique identifier, version of the protocol, etc. Figure 3 shows the definition described above.

```
module ROMIOP {
    typedef octet          PacketType;
    const   PacketType     Request = 0;
    const   PacketType     Reply = 1;
    const   PacketType     ACK = 2;
    const   PacketType     NAK = 3;
    const   PacketType     KeepAlive = 4;
    const   PacketType     Cancel = 5;
    const   PacketType     MemberChange = 6;
    const   PacketType     MsgOrder = 7;

    struct PacketHeader_1_0 {
        char           magic[4];
        octet          version;
        octet          flags;
        short          packet_type;
        unsigned long  packet_size;
    };
}
```

Fig. 3. ROMIOP packet definition

3.3 Proposed Extensions for Integrating ROMIOP and ETF

Since, until the moment, the ROMIOP specification is still not finished, several considerations had to be taken and added so that the implementation of this specification became achievable. The most important one was the creation of a new type of message, the one who carries the messages orders sent by the leader server (`MsgOrder`). Its definition is very simple: there is an identification of who sent the message and a list with a structure, which was also created, that contains the identification of the message and the identification of the member who sent the message.

The message order to be delivered is exactly the same of the list. It is necessary to send both the message identification and the member who sent it, because in the cases were there are messages with the same identification, the member who has the lesser identification will have its message delivered first.

Another crucial point is related to the reply consolidation. It is only said which models must be implemented (simple or quorum voting), however it is not informed where the consolidation must occur (in the sender or receivers). In this implementation, the consolidation algorithm is processed in the client who has made

the request, taking out of the servers this extra load of processing. This issue could be considered the most laborious to be developed.

Since the fact that there is not any module related with the reply consolidation in the ETF specification, as well as a lack of related work, the entire model had to be developed to solve the problem. Several approaches were analyzed, implemented and tested before reaching the definitive method that attended the incomplete ROMIOP specification. Because of this fact that the protocol did not use as base any other existent protocol (see figure 1 at section 3). The control of the received replies is not of the protocol. The middleware is the only one who defines which process (client) owns the reply. This approach had to be detoured, since with this, only the first reply would always be used, getting rid of all the others (this fact would always occur in the existence of more than one server). The chosen solution was to create a “deviation” of this information to the ROMIOP protocol, after the one being sent to the middleware, attending this way both specifications (ETF and ROMIOP).

Although it may look a slow method (the same information passes through the protocol twice) the tests that were made (see section 6) prove that the performance loss was insignificant. Beside that, this method has the advantage of modularizing the functions: messages that have to be consolidated go through this process, while the ones that do not need, follow a direct path. With this, each client process the consolidation of its requests while the servers stay with the task of replying the requests and defining the order of the messages to be delivered.

Exactly because of the need to consolidate the replies that were implemented a member service (membership) with more functionality than the one already existent in MJaco. The protocol needed to know exactly the quantity of functional members so that the client knows how many replies it must wait for. This entire module was projected without any kind of specification. Since the protocol needs to keep an updated list with the members, and it must also be capable of handling omission faults, crash faults and problems with the physical network (like a cut network cable or a badly configured router), it was implemented an algorithm that is executed by the leader server. With it, at each pre-determined time interval, a message is sent to every member of the group asking if they are still alive. All the members that do not send an ACK to this message will be removed from the group, in other words, it is assumed the perfect detector abstraction.

Finally, the specification in its current state does not define the total ordering algorithm to be used. With this, it was defined an algorithm based on the fixed sequencer [5], presented in section 4, so that it became possible to verify the potentialities of the proposed architecture.

4 Atomic Multicast Algorithm

A total order algorithm with reliable multicast is the one that guarantees that all non-faulty members of a group will deliver the same set of messages in the same order [5]. This type of algorithm is also called atomic multicast, because the delivery of a message happens as an indivisible primitive: the message is either delivered to everyone or to no one, and if delivered, all the other messages will be ordered either before or after this one.

This type of ordering makes easy the maintenance of a consistent global state between several processes, being used as a base to the implementation of fault tolerant through the active replication (the current state is replicated) [19].

It is important to note the way that the packet unique identifier that each member puts in every message sent to the group is used. This ID is nothing more than a local counter that is started with the number zero. Every sent packet increments this counter. Also, every packet that any member receives, even if is not directed to him, the ID is analyzed. If it is bigger than its local counter, than this value becomes its local counter number. If it is the same or less, nothing is done. With this it is possible to use the classic algorithm of events ordering of [9], which allows the identification of the order in local messages (the message with the lesser identifier will always be delivered before the message with the bigger identifier).

4.1 Assumptions and System Model

Regarding the process failures, we assume a crash fault model. The group service module tolerates processes faults since it keeps an updated list of members.

We assume reliable channel, this semantic is implemented by the periodic retransmission of messages until every receiver processes acknowledge the receipt of the message (through an ACK message). Duplicated messages are detected by its identification and are discarded, however ACKs to these messages are sent before the discard occurs, since its receiver may lose the ACK.

Assumptions regarding time had to be made to implement the ROMIOP protocol. To determine the re-send of the messages because of the non-receipt of enough ACKs was adopted that the sum of the times of computing and communication are synchronous, in other words, there is an upper limit known so that both occur. Another presumption taken regarding time was of a perfect failure detector. If the time taken to the reply of the message asking if it is alive (KeepAlive) arrives after a certain value (or never arrives), that process will be considered as being with problem and will be excluded from the group.

If any process locks and do not return (crash) there will be no problem, since the membership module keeps a list of the quantity of members updated. Another possibility is the lock of a process and its return after a period of time. In this case, depending of the time it stays without answering it may be removed from the group, however it will be re-included in the group members when it returns. The only problem related to this last possibility is that the receivers will discard any reply, from the excluded member, originated from a requisition delivered before the lock happened.

Finally, it was implemented a leader election algorithm in which the first process that enters the group is considered the leader. This functionality was implemented by simply sending a message to the group and waiting for an answer by a determined period of time. If no one replies, it considers itself as the leader and starts to warn every other process that enters the group of the existence of a leader. Notice that every timeout parameter, related to these times, are configurable so that the protocol can work in the best possible way in each network environment.

4.2 Algorithm

The ROMIOP adopted protocol for atomic multicast is based on the fixed sequencer paradigm [5]. Basically, the protocol works in the following manner: the emitter casts a message requesting the members of the group to enter this group. Next, it can send

```

1.  (Initialization)
2.  buffer <- φ (received messages buffer)
3.  received <- φ (received and not delivered requests buffer)
4.  order <- φ (order messages of request to be delivered buffer)
5.  members <- myself.id (group members list)

6.  (To R-multicast(m)) (group message diffusion)
7.  if m.type = REQUEST | REPLY | MEMBERCHANGE | MSGORDER then (m.type:
   type of message m)
8.    buffer <- buffer U {m}
9.    timeout(m, 2000) (start timeout to re-send if needed)
10.   send(m) (send the message)
11. end if
12. else if m.type = ACK then
13.   send(m)
14. end if

15. (To R-receive(m)) (receipt of a message)
16. if m.type = REQUEST then
17.   received <- received U {m}
18. end if
19. else if m.type = REPLY then
20.   consolidate(m) (consolidate the received replies)
21. end if
22. else if m.type = MEMBERCHANGE then
23.   if m.status = added then (m.status: type of member change)
24.     members <- members U {m.id}
25.   end if
26.   else if m.status = removed then
27.     members <- members \ {m.id}
28.   end if
29.   else if m.status = online then
30.     R-multicast(members)
31.   end if
32. end if
33. else if m.type = MSGORDER then
34.   order <- order U {m}
35.   while m.id ∈ received.id do
36.     deliver(received) (deliver the message in the correct
   order)
37.     order <- order \ m
38.     received <- received \ m
39.   end while
40. end if
41. else if m.type = ACK then
42.   if enoughACK(m) = OK then (if received an enough number of
   ACKs)
43.     timeout(m).stop (stops the timeout that re-sends the
   message)
44.     buffer <- buffer \ m
45.   end if
46. end if

```

Fig. 4. ROMIOP simplified algorithm

message requests to the group address and it stays waiting for a quantity of ACKs equal to the quantity of receivers in the group. If the quantity of ACKs is lesser than the expected, the request is re-send to the group. The ROMIOP simplified algorithm is shown in figure 4.

Initially, all buffers are initialized with empty values (line 1 to 5). To multicast a message to the group (line 6) they need first to be stored (line 8) in a local buffer (excluding the ACK type of message). After the message is stored, a timer is created (line 9). Just after both these steps that the message is sent (line 10). This is needed because the messages have to receive confirmations (ACKs) that it effectively reached its destination. If an enough number of confirmations do not come, the message is re-send after the finish of the timer. Upon the receipt of ACKs (line 41), if they are sufficient (line 42), the timer to that message is stopped (line 43) and the buffer that stored the message is deleted (line 44).

Every received message has a different type of processing (line 15). The requests (line 16), only received by servers, are stored in a local temporary buffer (line 17). These requests are only delivered upon the reach of the message with the order of the messages (line 33). After the receipt of the order, the servers compare its identifications with the ones that they have stored in its local buffer of already received messages (line 35). All messages that it has, starting from the first one and going sequentially to the last, will be delivered (line 36). If, by any reason, the server does not have one of the messages contained in the order, all the subsequent messages will not be delivered until the missing one is received.

The leader server sends the message with the order of the messages (MsgOrder type). Always after receiving a request message, a configurable timer is started. After the end of this timer, it is sent to the other servers belonging to the group the order message, containing the identification of all received requests messages during that time.

Just after the processing of the request message by the server, if it needs a reply, a reply message type will be sent to the group address (line 6). The emitter client who sends the request message stays waiting for a certain number of reply messages, depending on how the consolidation was configured (line 19 and 20).

Finally, every time an object wants to enter the group, it sends a message of type MemberChange with the status added. All participants of the group that receive this kind of message (line 23) add the member who sent the message to its local list. After the member receives the confirmation that he entered the group, it sends another MemberChange type of message, but with the online status. When the members receive this message (line 29) they reply sending its local list of members.

5 Implementing ROMIOP Below ETF

To better present the protocol and the way it was adopted to be implemented, in order to satisfy the requirements of the ETF specification, follows the figure 5. It is a class diagram showing only the extremely essential methods and attributes, besides representing only the most important relationships between classes.

The section 3.1 already describes the steps to create the ClientROMIOPConnection and ServerROMIOPConnection classes. Both

are derived from the ROMIOPConnection class, being the first one created by the ROMIOPFactories while the second one by ROMIOPListener.

Basically, at the reception of any packet message, through the ROMIOPConnection class, the fragment, after several verifications, is added to an object of the FragmentedMessage class. When a message is considered complete it is then delivered to the ROMIOP algorithm, where it will be ordered.

For messages that need reply, the ReplyConsolidatorImpl and NotificationImpl classes are used, being the second controlled by the first, and this controlled by the ROMIOPStrategy class. The ReplyConsolidatorImpl effectively consolidates the reply and it is activated in the creation of the request message that needs the reply or in the receipt of any of its replies. The NotificationImpl class is only used to notify the ORB of the chosen reply.

Finally, the ROMIOPStrategy class keeps all the remaining necessary processes for the correct functioning of the protocol, being for this considered the most complex class and the one with more functionality. Between some of its functions are the send of ACKs, the membership control and the send of the message order.

Almost every protocol class use both MulticastUtil and ROMIOPProfile classes. Each one of them is responsible for storing specific information. The first one is related with the whole protocol, storing protocol configurations, like the consolidation method and the time limits. The second one is connected uniquely with a connection, being responsible for storing information of it, like the group address.

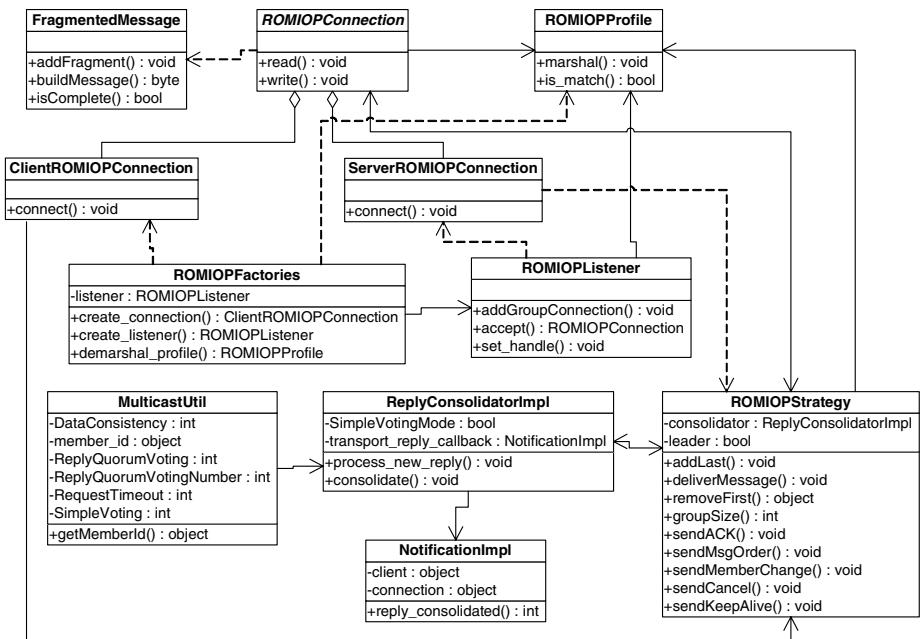


Fig. 5. ROMIOP plug-in simplified class diagram

6 Performance Evaluation

With the purpose of analyze the MJaco performance with the ROMIOP, as well as the choices made in the implementation of it, there were made several tests. The environment in which the tests were done was a set of machines running Windows XP as operational system. The client was executed in an Athlon 2600+ machine, with 512MB of RAM memory. The leader server was executed in a Pentium 4 at 2.6GHz with 1,5GB of RAM memory. The other two servers were executed in Athlon machines at 1.47GHz with 248MB RAM memory each.

The first test had the objective to analyze the scalability as well as the velocity of the algorithm. The principle is simple: a message with a variable size is sent and it stays waiting for a reply (an integer of 4 bytes). The final reply (the one that the client will actually use) is only consolidated after the receipt of all replies of each server (atomic type of consolidation). In figure 6 the result can be seen.

The result was exactly like the expected one. As much as the number of servers grown, the time that the client takes to consolidate the replies becomes bigger. Another point of interest is the low increase of cost with the insertion of more servers, proving that the adopted protocol is relatively scalable. It is important to say that the time the leader server takes to send the message order was configured to 10ms. Finally, it can be easily seen a risen in the time taken to accomplish the consolidation from 1 to 2 or 3 servers. This happens because the message order is not send with only one server.

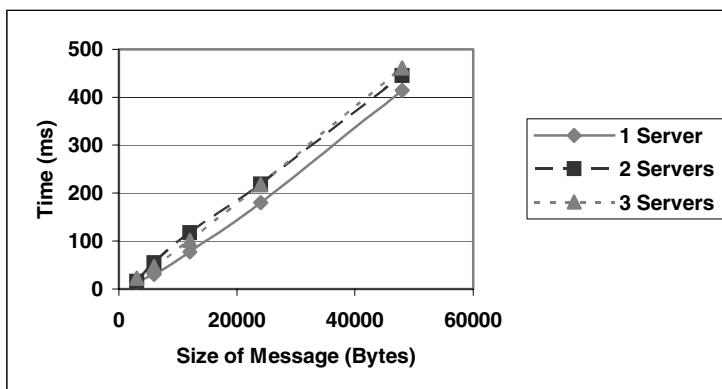


Fig. 6. Test result of the atomic consolidation

The following test was made to analyze the consolidation algorithm, where the atomic option was compared with the first reply option. The figure 7 shows the results for 2 and for 3 servers.

The analysis of the graphics brings, besides the expected that the time taken to consolidate only the first reply is smaller than the one with all (atomic), the fact that for small messages (approximately until 3000 bytes) there is practically no

performance improvement, being for this cases considered more advantageous to use the atomic consolidation, since it brings more security. With this result, it can be seen that for large types of messages it is crucial the choice of the consolidation method. If the application needs to support Byzantine faults or if you only desire to be sure of the obtained reply (like life support systems or military applications) there will be a considerable cost. Notice that even with the increase of the number of servers (from 2 to 3), the time to obtain the reply from the first reply consolidation method practically does not increase.

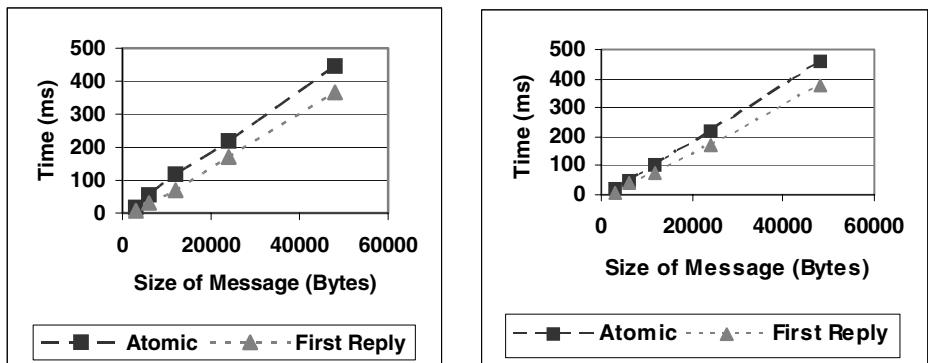


Fig. 7. Comparative with two types of consolidation methods with 2 and 3 servers

Next it was made a test to analyze one more consolidation option that ROMIOP implement. It was tested the percentage of members quorum system. The chosen value configured to make the consolidation was of 51% of members (figure 8).

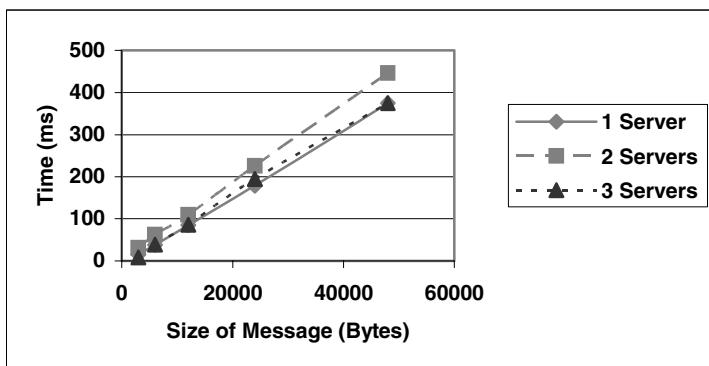


Fig. 8. Result of the consolidation by quorum test

The analysis of the obtained graphic shows that for this type of consolidation the cost for 2 servers is bigger than for 3, since with only 2 servers, all 2 have to give a

reply so that the consensus can be done. With 3 servers, only 2 of them (66%) have to give the answer so that the consensus can be done (if the answers given by both are the same). A very important thing to be noted is that the time to consolidate the reply with 1 and 3 servers is practically the same, proving that the chosen algorithm is fast and efficient.

The next test shows the time that the protocol took to only send a message of variable size without waiting for a reply (one-way). Figure 9 shows the obtained results.

With this result it can be analyzed the cost of not having been used any kind of flow control in the algorithm. Messages with a considerable size (approximately bigger than 40000 bytes) start to bring a loss of packets (UDP does not have any kind of flow control like TCP), being necessary to re-send some of the messages. The solution to this type of problem is relatively easy, since the ReMIOP protocol already addresses this issue. It is interesting to note that the quantity of servers practically does not influence in the time taken to send the message, which is perfectly correct, since the message is sent only once to all of them, via broadcast.

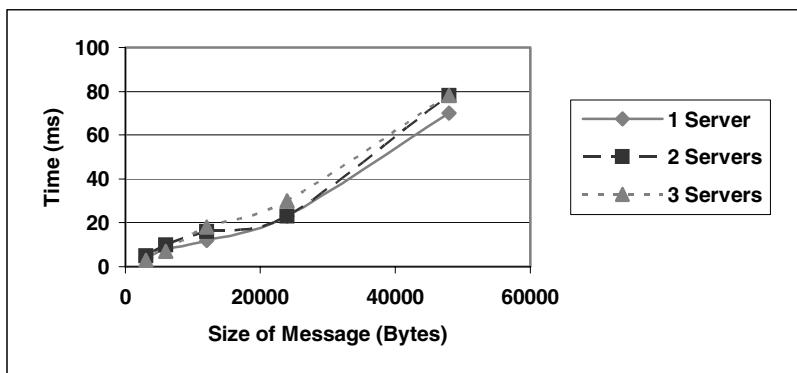


Fig. 9. Result of test without reply

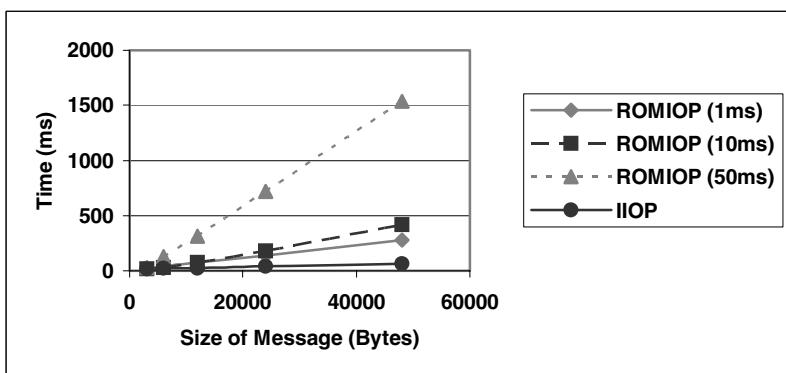


Fig. 10. Comparative of the total ordering cost

The following tests were done to compare the ROMIOP with the UMIOP, demonstrating the cost of implementing reliable multicast and total ordering. The first of these tests show the most important reason to the loss of performance, which is the total ordering. It was done by comparing the costs of invocation of a method with reply in a ROMIOP group with the same invocation being done in a normal object (without replication) accessed via IIOP. Figure 10 presents the results.

All servers must execute the request in the same order and the messages originated from older ones must be executed only after the one who generated it. To accomplish this, the servers must wait a time before sending the order of the messages to be sure that no older message will reach the servers, creating an erroneous order. This time is configurable in ROMIOP and it basically defines the protocol performance in requests where a reply is needed (two-way).

The result graph above compares the time taken to a system with only one server using a protocol that only provides point-to-point reliability and FIFO ordering (IIOP, which is the protocol that UMIOP [18, 1] uses for request that need reply) and the ROMIOP, with different times to send the message order. It is clear that any value above 10ms will slow down considerably the time taken to send the reply (look at the result with 100ms in the graph). Values below 10ms (look at the result with 1ms in the graph) brings very little benefits in terms of speed. This happens because the limiting factor starts to be not more the sent of the order message but the cost to accomplish the reliable multicast, creating the considerable difference to the IIOP.

Finally, the last test compares the performance of ROMIOP with MIOP [1] in requests without reply. Figure 11 shows the results obtained with 1 and 3 servers.

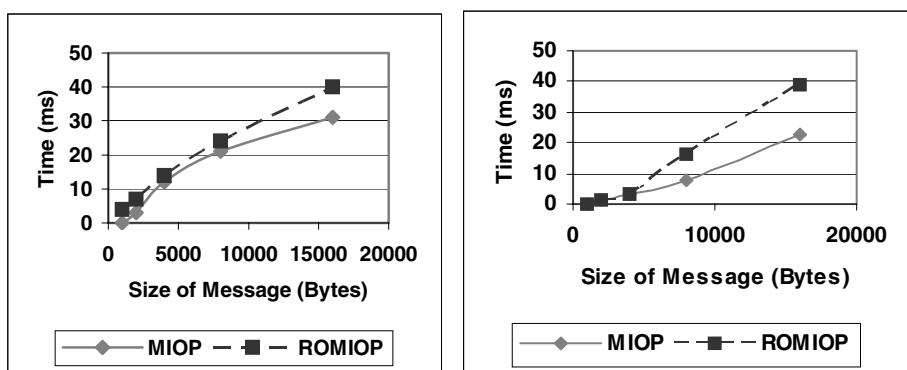


Fig. 11. Comparative of protocols with request without reply with 1 and 3 servers

The result obtained with this test revealed that for messages with small size (below approximately 5000 bytes) the performance of ROMIOP is really close to the MIOP [1]. Larger messages requires re-sending of some packets, making the difference grows (it is easier to visualize it with 3 servers).

7 Conclusion

This paper presented a study about the implementation of the ROMIOP draft specification using the principles of the ETF specification. The first specification aim to standardize interfaces and message formats for message multicast with total ordering and reliable multicast guarantees, while the second one defines methods for integrating new protocols into already existent systems.

One of the biggest problems we faced was the fact that the ETF specification does not support multicast communication and that it also does not easily allows to stack protocols, making the implementation of ROMIOP much more difficult and with the need to create several small extensions.

With the conclusion of this specification, we finally will have interoperable mechanisms for group communication in open (standardized) middleware. With ROMIOP now finished, it is possible to analyze each module in a detailed way, searching for a better performance, making the implementation more efficient and with more functionality.

More information related with the performance tests, the developed algorithm, as well as the source code can be found in the Internet, inside the web page of the developers group (<http://grouppac.sourceforge.net>).

References

1. Alysson Neves Bessani, Lau Cheuk Lung, Joni da Silva Fraga, and Alcides Calsavara. Integrating the Unreliable Multicast Inter-ORB Protocol in MJaco. Proc. of the 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems - IFIP DAIS'03, Lecture Notes in Computer Science vol 2893. Paris, France, 2003.
2. Alysson Neves Bessani, Joni da Silva Fraga, Lau Cheuk Lung, and Eduardo Adílio Pelison Alchieri. Active Replication in CORBA: Standards, Protocols and Implementation Framework. Proc. of 6th International Symposium on Distributed Objects and Applications – DOA'04, Lecture Notes in Computer Science vol 3291. Larnaca, Cyprus. October, 2004.
3. Alysson Neves Bessani, Joni da Silva Fraga, and Lau Cheuk Lung. Extending the UMIOP Specification for Reliable Multicast in CORBA. Proc. of 7th International Symposium on Distributed Objects and Applications – DOA'05. Lecture Notes in Computer Science (same volume). Larnaca, Cyprus. October, 2005.
4. S. E. Deering. Host extensions for IP multicasting. IETF RFC number 988. 1986.
5. Xavier Défago, André Schiper, and Peter Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comp. Surveys, 36(4):372-421, December, 2004.
6. Pascal Felber, The CORBA Object Group Service – A Service Approach to Object Groups in CORBA, PhD. Thesis, École Polytechnique Fédérale de Lausanne. 1998.
7. Pascal Felber, Benoit Garbinato, and Rachid Guerraoui. The Design of a CORBA Group Communication Service. In Proceedings of the 15th Symposium on Reliable Distributed Systems (SRDS'96), pages 150{159, Niagara-on-the-Lake, Canada, 1996.
8. Pascal Felber, and Priya Narasimhan. Experiences, Strategies, and Challenges in Building Fault-Tolerant CORBA Systems. IEEE Transactions on Computers, 53(5):497-511. 2004.
9. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 21(7):558-565. July, 1978.

10. Silvano Maffeis. Run-Time Support for Object-Oriented Distributed Programming, Ph.D. Thesis University of Zurich. 1995.
11. L. E. Moser, P. M. P. Melliar-Smith, and Priya Narasimhan. Consistent Object Replication in the Eternal System, Theory and Practice of Object Systems, 4(2): 81-92. 1998.
12. L. E. Moser, P. M. P. Melliar-Smith, Priya Narasimhan, R. R. Koch, and K. Berke. Multicast Group Communication for CORBA. In Proc. of International Symp. on Distributed Objects and Applications, pages 98{107, Edinburgh, United Kingdom. September, 1999.
13. Object Management Group. Object Management Group, Fault-Tolerant CORBA Specification v1.0. OMG Doc. ptc/2000-04-04. April, 2000.
14. Object Management Group. Reliable, Ordered, Multicast Inter-ORB Protocol. Initial Submission OMG Doc. realtime/2002-11-28. November, 2002.
15. Object Management Group. The Common Object Request Broker Architecture v3.0. OMG Standard formal/02-12-03. December, 2002.
16. Object Management Group. Extensible Transport Framework Specification v1.0. OMG TC Document ptc/2004-01-04. January, 2004.
17. Object Management Group. Reliable, Ordered, Multicast Inter-ORB Protocol. Revised Submission OMG Doc. realtime/2003-10-04. October, 2003.
18. Object Management Group. Unreliable Multicast Inter-ORB Protocol v1.0. OMG Doc. ptc/03-01-11. October, 2001.
19. Schneider, F. B. (1990) Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. ACM Computing Surveys, 22(4): 299-314. December, 1990.

The Design of Real-Time Fault Detectors

Serge Midonnet^{1,2}

¹ Ecole Supérieure d'Ingénieurs en Informatique et Génie des Télécommunications,
1 Rue du port de Valvins 77210 Avon, France

² Institut Gaspard Monge - Université de Marne la Vallée, 5, boulevard Descartes,
Champs sur Marne, 77454 Marne-la-Valle Cedex 2, France

Abstract. This paper presents the design and implementation of real-time fault detectors. We describe their design, implementation, and scheduling under a Fixed Priority/ High Priority First policy. Two types of real-time detectors are described; primary detectors and secondary (meta) detectors. A Primary Detector is designed for the detection of simple faults and failures (Worst Case Execution Time, Worst Case Response Time, Latest Response Time and Activation Overrun events). These events occur when a task uses more resources than have been catered for. The secondary type of detector, called *meta Detector*, is used to detect more complicated events called *meta-events*. Meta-events are based on a set of primary detectors and their interrelations. The Real-Time Specification Language (RTSL) is used for the description of Meta-events, including the primary events relations such as precedence; (THEN) and other logical relations; (AND, OR, TIMES). Primary and meta fault detectors must be admitted to the system as periodic or sporadic real-time threads. We present a method for the feasibility analysis of each detector type. These principles are integrated within a Minimum Real-Time CORBA prototype called RT-SORBET.

1 Introduction

Fault detection is used in order to prevent the tasks from missing their deadlines. We say that a task is faulty when it has been unable to complete within a specific time (i.e. the Latest Response Time, the Worst Case Response Time). A task is also faulty when it overuses the provisionned resource. A fault detector is attached to each Fault-Tolerant Real-Time thread for fault notification. A *temporal failure* occurs when a deadline has not been respected, a *temporal fault* occurs when another temporal characteristic (other than the deadline) has not been respected. For example, a fault may be said to occur when the worst case execution time of a task has overrun; when the fixed interval of a periodic event has overrun; or when the minimal inter-arrival delay of a sporadic event has overrun. Lack of monitoring of the above may result in a temporal failure, the consequences of which will affect both the faulty task and, more seriously, the other tasks.

In section 2 we describe briefly the requirements of both specifications (Real-Time CORBA and Fault-Tolerant CORBA).

In section 3 we introduce the RT-SORBET architecture for Fault Detection and the real-time application event model. In section 4 we describe the different types of real-time monitors supported. In this section, we also presents, the Real-Time Specification Language used for the specification of meta level real-time Fault Detectors.

In section 5 we describe the real-time scheduling model and the feasibility analysis method used for both, primary and meta real-time fault detectors admission. We conclude after a brief description of related works in section 6.

2 Fault-Tolerance and Real-Time Requirements

2.1 Real-Time CORBA Specification Requirements

The Real-time CORBA (RT-CORBA) standard [1],[2] describes different interfaces for application development, and mandatory properties, which an RT-CORBA system must respect. The most important property of an RT-CORBA system is the scheduling of execution entities (threads). The scheduling of threads is based on fixed priorities. The specifications detail how these priorities are set; either on the client side (Client propagated) or on the server side (Server-declared). Subsequently they will be translated into native priorities for the host operating system. End-to-end scheduling respects both thread priorities and thread deadlines. End-to-end predictability is achieved by setting and assuming bounds on message transmission latency across the network; and also by setting and assuming bounds on message processing time within the ORB.

2.2 Fault Tolerant CORBA Specification Requirements

The Fault-tolerant CORBA (FT-CORBA) standard [3] describes the architecture for the replication of CORBA objects. The essential operations an FT-CORBA system must carry out for both active and passive replication are; logging, checkpointing and message recovery; replicas management and distribution.

Strong replica consistency requires a deterministic behavior of objects.

An FT-CORBA system must guarantee on message transmission and delivery i.e. the same sequence of messages in the same order. There must be no loss of messages over the communication medium and no delivery of duplicate invocations or responses.

The system must also provide a state transfer mechanism for new and recovering replicas.

2.3 Fault-Tolerant and Real-Time Requirements

A real-time and fault-tolerant application requires one of the following in order to function:

- A special RT-FT scheduler. Real-time resource-aware scheduling function to detect faults and failures. Fault-aware function which decides when to initiate recovery after fault and failures notification.

- The ability to predict new resource allocations (cpu load, memory, network bandwidth), to control resource usage parameters (resource limits, current resource usage) and to perform proactive actions such as the migration of replicas to idle machines before the former execute. This situation occurs in the context of dynamic systems where new activities are scheduled online.
- The ability to predict faults and failures, the ability to trigger reactive action in response to overload conditions. The presented work focuses on this point.

3 Architecture for Fault Detection

With RT-SORBET [18] Real-time Servants are hosted as real-time threads (in a thread per servant operation policy [4]). These –time threads are triggered when a request is received. They process the Servant Operation instructions.

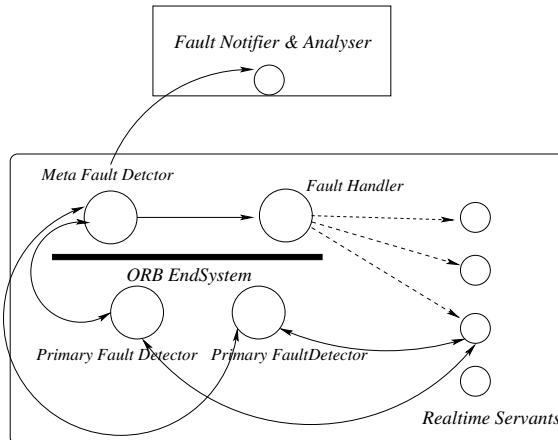


Fig. 1. Architecture for Fault Detection and Handling

An RT-SORBET architecture is composed of different types of objects; the real-time servant threads which must be monitored; the real-time detectors designed for monitoring specific events during the execution of their associated real-time Servant; the real-time fault handler which is activated when a fault occurs. A Fault Notifier and Analyser object may be used as a global or distributed oracle. What is more, in a distributed environment, interpretation may be quite different. For example, missing a deadline is considered as a local thread failure, but in the context of a distributed activity composed of multiple segments, a segment's missing a deadline does not necessarily lead to the end-to-end deadline being missed. In this paper we focus on the design, implementation and feasibility analysis of local fault detectors. These different elements of architecture are recalled in figure 1.

For a description of Fault handling especially in the case of overloaded systems, see [5] [6] [7].

The event model we are considering is the sporadic model. We define a sporadic task as follows:

A sporadic task τ_i consists of an infinite number of occurrences, whose request times are separated by a minimum time T_i termed *period*. Each occurrence of τ_i requires C_i *computation time* provided by the server and must be completed within a *deadline* D_i called the *relative deadline* of τ_i . If task τ_i is requested at time t , then $t+D_i$ is designated the *Absolute deadline*.

The sporadic model is well adapted to foreseeable activities (activities with a long duration of interactivity such as video conferences or telephone calls)

4 The Design of Real-Time Fault Detectors

A Fault Detector is made of two parts:

- a piece of code (the detector code) generally included in the standard real-time Java API or included in an extended package (javax.extended),
- a Fault Handler which is a sporadic real-time thread.

RT-SORBET proposes two levels of fault detectors:

- A primary level detector is devoted to monitoring a single event. Examples of primary fault detectors will be discussed later.
- A secondary or *meta level* of fault detector is associated generally to more than one primary detector and triggers its handler when one or more temporal or logical conditions occur.

4.1 Description of Primary Fault Events

In figure 2: a ↑ indicates task activation; a ↓ indicates task deadline; a ↗ indicates task response time; a ↘ indicates task latest response time. The overrunning of any of these timing constraints is a fault and there detection is required in order to prevent the failure of a non-faulty task. Detectors we take into account are:

Activation Overrun Detector. An *Activation Overrun* event occurs when a task is activated more frequently than expected. Its interarrival duration is shorter than the minimum period allowed for a sporadic thread (shorter than the period for a periodic thread).

Deadline Miss Detector. A *Deadline Miss* event occurs when a task deadline has been missed.

Response Time Overrun Detector. A *ResponseTime Overrun* event occurs when a task response time exceeds its expected value. The response time value of the task is processed at task admission using the feasibility analysis. The feasibility analysis is provided by the task scheduler. The worst case response time value of a real-time thread is processed using the method described in the section 5.2.

Latest Response Time Overrun Detector. A *LatestResponseTime Overrun* event occurs when a task response time exceeds the response time plus an extended value which we call the allowance value. The algorithm for the allowance value calculation is described in what follows.

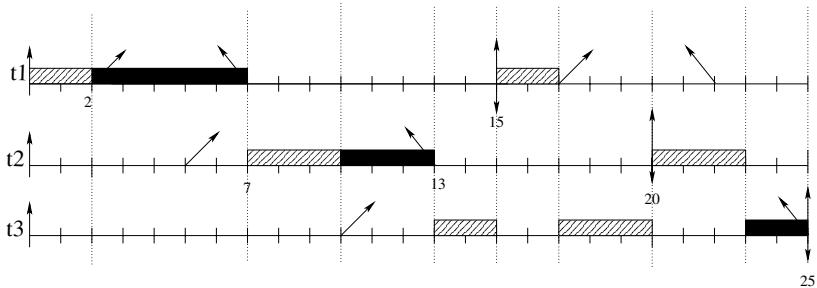


Fig. 2. Task execution with critical events

Figure 2 represents the execution of three periodic threads (τ_1, τ_2, τ_3) located on the same node. We observe their execution and critical events. For each thread the real-time parameters are; the thread Execution Cost (C_i); the thread Period(T_i); the thread Deadline (D_i); and the thread Priority (P_i). Other relative time values are processed during the thread admission phase, these values are; the thread Response Time (R_i); the thread Allowance (A_i); and the thread Lastest Response Time (LR_i). Non-faulty threads are represented by the grey boxes. The black boxes represents faulty threads executions. In this case each thread completes without missing its deadline.

Table 1 presents the critical events values for each thread.

Table 1. Critical Events Values

thread	Execution Cost	Deadline/Period/Priority	Response Time	Allowance	LatestResponseTime
τ_1	2	15/15/50	2	5	7
τ_2	3	20/20/40	5	8	13
τ_3	5	25/25/30	10	10	25

4.2 Design of Primary Fault Detectors

In this work, the monitored real-time tasks have been designed with both Sporadic and Periodic Java real-time threads. Some of the following Fault Detectors use direct detection by delegating detection directly to the JVM (Deadline Miss and ActivationOverrun detectors), others are indirect and need a specific thread monitor (ResponseTime and LatestResponseTime detectors). The implementation of real-time fault events is carried out using an *AsyncEvent* object. The sporadic thread (fault handler) associated with the event is an *AsyncEventHandler*

object. In order to trigger the event and then activate the fault handler (the sporadic thread) we use *fire()*, as the specific event object method is called.

Each detector has its own priority, inherited from the monitored thread, or set by the *SchedulingParameters* value. See [8] for a description of the Real-Time Specification for Java.

Design of a Deadline-Miss Detector. A thread deadline missed event can be detected directly by the real-time thread object. When this event occurs a *Deadline Miss Handler* is called (see the *AsyncEventHandler missHandler* parameter of the *SporadicParameters Class* constructor).

Design of an Activation Overrun Detector. When the minimum interactivation duration (*mit*) is not respected the policy for dealing with minimum arrival time violations must be one of the following:

1. The activation is ignored (*mitViolationIgnore* value)
2. An exception is called (*mitViolationException* value)
3. The new release replaces a previous release (*mitViolationReplace* value).
4. There will be no checks for correct minimum interarrival times for incoming releases (*mitViolationSave* value).

We use the *setMitViolationBehavior* method of the *SporadicParameter* object to setup the detection of the Activation Overrun event. The *mitViolationException* object triggers the *AsyncEvent(Fault)Handler*.

Design of ResponseTime Overrun Detector. In order to implement the detection of that type of event the standard RealTime Thread Class has been extended. The detector is a *OneshotTimer* set with a value which is made up of the current time plus the worst case response time of the monitored thread. The *OneShotTimer* is setoff by each activation thread. The *OneShotTimer* when executed tests the completion of the monitored thread. If the monitored thread is not completed then the *OneShotTimer* triggers the Fault Event.

```

1.      For each t in TaskSet Do
2.          If (k ≥ 1)
3.              allow ← 0
4.              Do
5.                  allow ← allow + 1
6.                  t.Ci ← t.Ci + 1
7.                  For each j in FaultTolerantTaskSet(t)
8.                      j.Ci ← j.Ci + 1
9.              While(ifFeasible)
10.                 t.Ai ← -1
11.                 ResetCi(t,allow)
12.

```

Fig. 3. ResponseTime Allowance algorithm

Design of LatestResponseTime Overrun Detector. The calculation of the *Allowance* value for each task is carried out incrementally by adding a time unit to each task cost as long as the hole remains feasible. In this way the available resources are distributed fairly, see figure 3 for algorithm description. This value is calculated by the *addToFeasibility* method of the extended RealTime Thread class. The detector design is the same as in the preceeding case (using a OneShotTimer).

4.3 The *Meta Fault Detectors* and Their Specification Language, RTSL

A *meta Fault Detector* (see figure 4) is used to detect more complicated events called *meta-events*. Meta-events are based on a set of primary detectors and their interrelations. Thus *meta Fault Detector* detects compound faults and triggers a related *meta Fault Handler*.

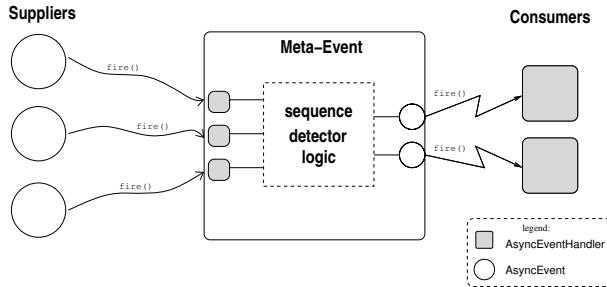


Fig. 4. Meta Fault Detector Architecture

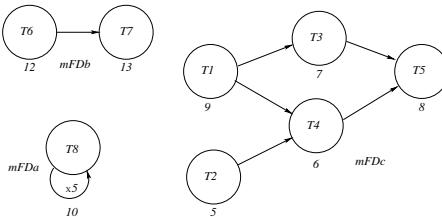


Fig. 5. Event Relations in a Secondary Fault Detector

We have designed a language (*RTSL*) and a compiler *RTSLcomp* to facilitate the implementation of the meta Detectors. The generated code is based on the Real-Time Specification for Java.

Using *RTSL* Suppliers are defined as primary fault detectors and Consumers as meta fault handlers. In order to specify a *meta Fault Detector*: Firstly we

define the types of *Suppliers* and *Consumers* we use. Then we define each meta Fault Detector using the keyword *MetaEvent*. A MetaEvent contains an *Event-Condition* keyword used to describe the relations between primary events. The current version (1.0) of RTSL supports only three logical relations (AND, OR, TIMES) and the precedence relation (THEN). In the following exemple (see figure 5) we have to monitor three applications. Application 1 is made of a single thread (T8), application 2 is made of 2 threads (T6 T7), application 3 is made of five threads (T1, T2, T3, T4, T5). We define three *meta Fault Detectors* (*mFDa*,*mFDb*,*mFDc*) in order to monitor the preceeding applications.

```

Events {mFDa,mFDb,mFDc};

Suppliers {LatestResponseTimeOver,ActivationOver};
Consumers {Finalize,Stop};

MetaEvent mFDa{
    Supplier LatestResponseTimeOver:T8;
    Consumer Finalize:F1;
    EventCondition T8 TIMES 4;
};

MetaEvent mFDb{
    Supplier LatestResponseTimeOver:T6,T7;
    Consumer Stop:C2, Finalize:F2;
    EventCondition T6 THEN T7;
};

MetaEvent mFDc{
    Supplier LatestResponseTimeOver:T1, T2, T3, T4, T5;
    Consumer Finalize:F3;
    EventCondition (T1 AND T2) THEN (T3 AND T4) THEN T5 ;
};

```

5 Scheduling and Feasibility Analysis of Real-Time Detectors

5.1 The Scheduling of Real-Time Detectors

Scheduling theory applied to hard real-time systems has been widely studied in the last twenty years. A lot of results have been achieved much progress have been made in th field of non-idling scheduling over a single processor. The most effective real-time schedulers make use of the HPF (Highest Priority First) on-line policy, two approaches are identified: fixed priority driven schedulers and dynamic priority driven schedulers. For fixed priority driven schedulers, refer to [9], [10] and [11]. Different scheduling is investigated for periodic or sporadic

tasks, depending on the relation of the period and the relative deadline of a task. We can identify three scheduling policies: Rate Monotonic, Inverse Deadline Monotonic (IDM) or arbitrary. Fixed priority driven schedulers were chosen for easy implementation. Most of the RealTime Java Environment offers only an FP/HPF scheduler. For dynamic priority driven schedulers, we can identify (not exhaustive) Shortest Slack Time (SST), Earliest Deadline First (EDF) and First In First Out (FIFO). Dynamic priority driven schedulers are considered better theoretically than fixed priority schedulers [12]. An EDF scheduler has been developped for the *Esmertec/Jbed* real-time Java Environment and another has been developed for the *MIT/Flex* real-time Java Environment.

In this work and in the current implementation of the RT-SORBET framework we have chosen to focus on FP/HPF. FP/HPF is the pre-emptive version of Fixed Priority/Highest Priority First non-idling scheduling. Bearing in mind that FP/HPF schedules the tasks according to their priority: the task with the highest priority first.

5.2 Feasibility Analysis for *Primary Fault Detectors*

Primay Fault Detectors are independent tasks. Their feasibility analysis depends on their priority. We use two priority classes, one for application real-time threads and one for fault detectors. Fault Detectors threads have higher priorities than application real-time threads. The priority of a Fault dectector is determined by $\phi_i = P_i + \text{Offset}$, where P_i is the priority of the monitored application thread. The smaller detector priority is greater than the higher application thread priority.

Definitions.

- *An idle time t* is defined on a processor as a time where there are no tasks released before time t pending at time t . An interval of successive idle times is called an idle period.
- *A busy period* is defined as a time interval $[a,b)$ where there is no idle time in $[a,b)$ (the processor is fully busy) and such that both a and b are idle times.
- *The processor utilization factor (U)* is defined as the fraction of processor time spent in the execution of the task set (see [9]). An obvious necessary condition for the feasibility of any task set is $U \leq 1$ (this is assumed in the sequel).

$$U = \sum_{t=1}^n \frac{C_t}{T_t} \leq 1 \quad (1)$$

The following condition 2 is a necessary and sufficient condition for the theoretical feasibility analysis of a realtime thread system. In the general case the deadline of a task can be greater than its period. For simplification of the metaDetector analysis, we don't authorize the reentrance of the primary detectors, i.e. a new fault handler can't start before the previous fault handler completes.

In the following; the worst case response time is found in the first period (in the synchronous task activation scenario).

$$r_i = \max_{q=0 \dots Q} (r_{i,q} - qT_i) \quad (2)$$

where Q is the number of periods within the Busy Period. where $r_{i,q}$ is the response time of period q .

Lehoczky and al. [11] have shown that the preceding equation ends when:

$$r_{i,q} \leq (q + 1) \times T_i$$

$$r_{i,q}^{n+1} = (q + 1)C_i + \sum_{j \in HP(i)} \left\lceil \frac{r_{j,q}^n}{T_j} \right\rceil \times C_j \quad (3)$$

5.3 Feasibility Analysis for *Meta Fault Detectors*

In this section we analyse the feasibility of the admission of a *meta Fault Detector*. Bearing in mind that a *meta FD* is made up of a set of realTime threads. The admission of a *meta FD*, requires the analysis of its influence on the response time of other tasks in the system and the analysis of the influence of the other *meta FD* on its own response time. We characterize the meta FD with the determination of its worst case response time (the response time of the last thread in the graph). precedence constraints. The worst case situation we must analyse is the situation where the load due to the detector activity is the highest.

This situation occurs when the sporadic event model becomes periodic. In this case the inter-event notified is the minimum period of the monitored real-time thread.

In this situation we can calculate the response time of the *meta FD*. In other cases the load is lower and the system remains feasible. The response time has no highest limit in the case of a sporadic model, as some events would never be notified.

The lowest response time of the sporadic event traffic model is the worst case response time of the periodic event traffic model. To find the *lowest sporadic response time* of the Meta detector we must process its *worst case periodic response time*. In this section we analyse the minimum response time of a Meta Fault Detector. The minimum response time corresponds to the shortest delay between the notification of the first event in the graph and the triggering of the real-time fault handler. The minimum value we can get is found when all the events are fired at their highest frequency. In this case the event model becomes a strict periodic model where events are fired at each detector activation.

The logical relations (AND, OR, TIMES).

- AND:

All the events must be notified but no precedence constraint has been defined. The Response Time of the metaFD i (R_i^{metaFD}) is the sum of the independent threads response times.

$$R_i^{metaFD} = \sum_{j \in FDThreadSet} r_j \quad (4)$$

Where r_j is processed using equation (2).

– OR:

One of the events must be notified.

$$R_i^{metaFD} = \max_{j \in FDThreadSet} r_j \quad (5)$$

– TIMES

$$R_i^{metaFD} = r_i \times p \quad (6)$$

Where p is the number of occurrences of the fault.

The precedence relation (THEN). We use the theoretical results from Harbour and al. [13] to determine the influence of the precedence constraints for the feasibility analysis of *meta Fault Detectors*. We assume that:

- Threads in a *meta FD* graph have the same period (the same period as the monitored real-time thread).
- A thread is not reentrant i.e. thread instances are scheduled in a FIFO order, the instance $(k+1)$ of a thread could not be influenced by the preceding instance k .
- Fault Detectors are not reentrant i.e. a new instance of a graph cannot start before the current instance completes (any thread in a graph could not be influenced by any other preceding thread in the same graph).

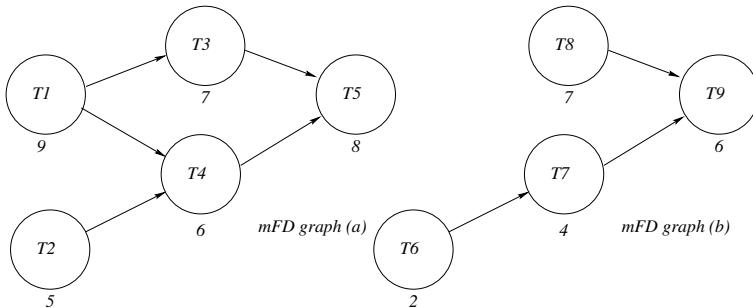


Fig. 6. Event Relations in Secondary Fault Detectors

We study the worst case response time of the thread τ_5 of the meta FD graph (a) (see figure 6). The number under each graph node is the task priority.

Each graph will be modified through several steps [13]: We describe the transformation of the graph (a).

1. *The graph is transformed into a chain.*

We look for the threads without any predecessor or with any predecessor not already placed (τ_1, τ_2), we place in first position in the chain the thread with the highest priority (τ_2).

$$\tau_2(5) \quad (a)$$

Then we continue with the thread with any predecessor not already placed. The only solution is (τ_1).

$$\tau_2(5) \prec \tau_1(9) \quad (a)$$

Then (τ_3, τ_4).

$$\tau_2(5) \prec \tau_1(9) \prec \tau_4(6) \quad (a)$$

We end with (τ_3 and τ_5).

$$\tau_2(5) \prec \tau_1(9) \prec \tau_4(6) \prec \tau_3(7) \prec \tau_5(8) \quad (\text{chain } a)$$

The result for the transformation of the graph (b) is:

$$\tau_6(2) \prec \tau_7(4) \prec \tau_8(7) \prec \tau_9(6) \quad (\text{chain } b)$$

2. *The chain is transformed into a canonical form.*

This transformation consists in the modification of the thread priorities. For each thread in the chain we look for the priority of its predecessor. If the priority of its predecessor is higher, it is changed to the current thread priority. And this recursively until the beginning of the chain.

The preceeding chain transformation:

$$\tau_2(7) \prec \tau_1(9) \rightarrow \tau_2(9) \prec \tau_1(9)$$

$$\tau_2(9) \prec \tau_1(9) \prec \tau_4(6) \rightarrow \tau_2(9) \prec \tau_1(9) \prec \tau_4(6)$$

$$\tau_2(9) \prec \tau_1(9) \prec \tau_4(6) \prec \tau_3(7) \rightarrow \tau_2(9) \prec \tau_1(9) \prec \tau_4(7) \prec \tau_3(7)$$

$$\tau_2(9) \prec \tau_1(9) \prec \tau_4(7) \prec \tau_3(7) \prec \tau_5(8) \rightarrow \tau_2(9) \prec \tau_1(9) \prec \tau_4(8) \prec \tau_3(8) \prec \tau_5(8)$$

and the canonical form of the chain mFD (a) becomes:

$$\tau_2(9) \prec \tau_1(9) \prec \tau_4(8) \prec \tau_3(8) \prec \tau_5(8) \quad (\text{canonical chain } a)$$

the canonical form of the chain mFD (b) becomes:

$$\tau_6(7) \prec \tau_7(7) \prec \tau_8(7) \prec \tau_9(6) \quad (\text{canonical chain } b)$$

3. *The thread sequence identification.*

In the preceeding exemple, we obtain one ore more sequences of threads with the same priority. We found the sequences:

- for the chain mFD(b) we found: The sequence $\sigma_{1,1}$ of priority 9 is composed of threads τ_2 and τ_1 and the sequence $\sigma_{1,2}$ of priority 8 is composed of threads τ_4 , τ_3 and τ_5 .

$$\sigma_{1,1}(9)(\tau_2, \tau_1); \quad \sigma_{1,2}(8)(\tau_4, \tau_3, \tau_5) \quad (\text{chain sequence } a)$$

- for the chain mFD(b):

$$\sigma_{2,1}(7)(\tau_6, \tau_7, \tau_8); \quad \sigma_{2,2}(6)(\tau_9) \quad (\text{chain sequence } b)$$

We can now represent a *meta FD* as a simplified system $S(\sigma_i)$ composed of *meta threads* (thread sequences). The execution time C_{σ_k} of a sequence σ_k becomes:

$$C_{\sigma_k} = \sum_{j \in \sigma_i} C_j \quad (7)$$

4. The determination of the Worst Case Response Time a thread τ_i

In the current exemple we determine the worst case response time of the meta FD (a) i.e. the worst case response time of the thread τ_5 in the same graph.

(a) Classification of thread sequences (HMP/HSB).

We must take into account the influence of other *FD* thread sequences on the thread in question. The other thread sequences are typed (H and L types). An H sequence is made of threads with priority equal or greater than τ_i and an L sequence is made of threads with priority lower than τ_i .

Several effects may influence the response time of τ_i . The effect of other graphs sequences may be a *Blocking Effect* or a *Preemptive Effect*.

A *Blocking Effect* occurs when an H sequence is preceeded by an L sequence, the H sequence cannot be executed unless the L sequence finishes. Harbour in [5] shows that in this case the H sequence can be postponed once. We call this type of sequence an *HSB* sequence i.e. High Simple Blocking sequence.

A *Preemptive Effect* occurs when an H sequence has no L sequence as predecessor. This H sequence may be activated several times in the present thread's σ_2 Busy Period. We call this type of sequence an *HMP* sequence i.e. High Multi Preemptive sequence.

In the exemple of figure 6 there is, in regard of task τ_5 a HMP sequence (thus a Preemptive Effect) and no HSB sequence (no Blocking Effect).

$$\tau_6(7), \tau_7(7), \tau_8(7), \tau_9(6) \quad (\text{the HMP sequence})$$

(b) Determination of the ϕ_{τ_i} -level Busy Period.

The concept of a **Busy Period** was first introduced in real-time scheduling by Lehoetzky [11] and modified by Harbour and al [13] to accommodate the fact that a single task may have subtasks with different priorities. The worst case response time of a task τ_i with a priority ϕ_i is found in a ϕ_i -level busy period. We must first determine the length of the ϕ_i level Busy Period.

The length of the ϕ_i busy period is determined by considering:

- the multiple preemptive tasks relative to the first subtask of the canonical chain (a)(MP_{i1}),
- the single preemptive tasks relative to the first subtask of the canonical chain (a)(SB_{i1}),
- and the complete instance of τ_i .

$$L_i = \min(t > 0 \mid \sum_{\tau_p \in MP_{i1}} \left\lceil \frac{t}{T_p} \right\rceil \times C_p + \sum_{\tau_p \in SB_{i1}} C_p^h + \left\lceil \frac{t}{T_i} \right\rceil \times C_i = t)$$

The number N_i of instances in the Busy Period is found thus:

$$N_i = \left\lceil \frac{L_i}{T_i} \right\rceil$$

- (c) *Step 3: Check the completion time of each of the N instances in the ϕ_i level busy period.*

The procedure then determines the completion time of the first subtask of the transformed canonical form. It then determines the completion time of the (j+1) subtask as a function of the j subtask until the completion time of the final subtask has been determined. This is performed for every job in the busy period.

- i. The completion time of the first thread(subtask) $\tau_{i,1}$ of the instance k of the graph τ_i (in its canonical form) is represented by $E_{i,1}(k)$.

$$E_{i,1}(k) = \min(t > 0 \mid \sum_{\tau_p \in MP_{i1}} \left\lceil \frac{t}{T_p} \right\rceil C_p + \sum_{\tau_p \in SB_{i1}} C_p^h + (k-1)C_i + C_{i,1} = t)$$

The completion of this subtask is influenced by; the blocking term, all higher priority processing initiated at the same instant and the execution time of the subtask itself.

- ii. Given the completion time of the first thread of the instance k, it is possible to calculate the completion time of the second thread of the instance k of τ_i .

$$E_{i,2}(k) = \min(t > 0 \mid E_{i,1}(k) + MP_{i2}Effect + SB_{i2}Effect + C_{i,2} = t) \quad (8)$$

where

$$MP_{i2}Effect = \sum_{\tau_p \in MP_{i2}} \left[\left\lceil \frac{t}{T_p} \right\rceil - \left\lceil \frac{E_{i,1}(k)}{T_p} \right\rceil \right] C_p$$

$$SB_{i2}Effect = \sum_{\tau_p \in SB_{i2}} \min(1, \left[\left\lceil \frac{t}{T_p} \right\rceil - \left\lceil \frac{E_{i,1}(k)}{T_p} \right\rceil \right]) C_p^h$$

- iii. In general, given the completion time of the j thread of the instance k , it is possible to calculate the completion of the $(j+1)$ thread of the instance k of τ_i .

$$\begin{aligned}
 E_{i,j+1}(k) &= \min(t > 0 \mid E_{i,j}(k) + MP_{i,j+1}Effect + \\
 &\quad SB_{i,j+1}Effect + C_{i,j+1} = t) \\
 MP_{i,j+1}Effect &= \sum_{\tau_p \in MP_{ij+1}} \left[\left\lceil \frac{t}{T_p} \right\rceil - \left\lceil \frac{E_{i,j}(k)}{T_p} \right\rceil \right] C_p \\
 SB_{i,j+1}Effect &= \sum_{\tau_p \in SB_{ij+1}} \min(1, \left[\left\lceil \frac{t}{T_p} \right\rceil - \left\lceil \frac{E_{i,j}(k)}{T_p} \right\rceil \right]) C_p^h
 \end{aligned} \tag{9}$$

The Worst Case Response Time of τ_i becomes:

$$R_{\tau_i} = \max((k-1) \times T_i - E_{im(i)}(k)) \geq 0 \text{ with } k \leq N_i \tag{10}$$

6 Related Works

The MEAD (Middleware for Embedded Adaptive Dependability) system [14] attempts to reconcile the conflicts between real-time and fault-tolerance properties in a resource-aware manner. One novel aspect of MEAD is its use of a proactive dependability framework that lowers the impact of faults on a distributed application's real-time schedule. The aim here is to design and implement mechanisms that can predict, with some confidence, when a failure might occur, and that can compensate for the failure even before it occurs.

The ROAFTS project [15] (Real-time Object-oriented Adaptive Fault Tolerance Support) is a middleware architecture designed to support adaptive fault-tolerant execution. While ROAFTS contains fault tolerance schemes devised for quantitatively guaranteed real-time fault tolerance, it is also designed to relax those characteristics while the application is in a soft real-time phase in order to reduce resource use.

The Time-Triggered Architecture (TTA) [16] and [17] is a distributed computer architecture for the implementation of highly dependable real-time systems. A TTA system has fault tolerance implemented in both hardware and software. In a TTA system, the membership and the clique avoidance algorithms detect state inconsistencies and force the nodes to restart, if their state is different from that of the other nodes.

The NEXT TTA project (High-Confidence Architecture for Distributed Control Applications) enhances the structure, functionality and dependability of the time-triggered architecture (TTA). Event-triggered communication services are integrated into the TTA to increase the required flexibility. The synchronous programming environment LUSTRE and its tool set are extended for the TTA and automated worst-case-execution-time analysis is explored. CORBA compliant interfaces are provided in order to make TTA systems interoperable with an open information infrastructure.

7 Conclusion

This paper offers both a theoretical and a practical approach of the real-time fault detection problem. The practical approach is centred on the design of primary and meta real-time fault detectors using the RealTime Specification for Java. The theoretical approach using classical real-time scheduling results is centred on the feasibility analysis of fault detectors. We offer a formalization of more complex real-time detectors with the RTSL language. An important highlight of the RTSL framework is that it provides a code generation in order to facilitate meta FD implementations. The Meta-Fault Detector framework have already been integrated into the RT-SORBET Environment in order to monitor local threads. The next step in this work will be the design of distributed Meta Fault Detectors and Global Notifiers.

References

- [1] Object Management Group Real-time CORBA Specification - Dynamic Scheduling, OMG Document formal/03-11-01 version 2.0, November 2003
- [2] Object Management Group Real-time CORBA Specification - Static Scheduling, OMG Document formal/05-01-04 version 1.2, January 2005
- [3] Object Management Group CORBA Fault Tolerant formal/04-03-21 chapter, v3.03
- [4] D.C. Schmidt, "Evaluating Architectures for Multi-threaded CORBA Object Request Brokers," Communications of the ACM Special Issue on CORBA, vol. 41, no. 10, Oct. 1998.
- [5] G.C. Buzzato, J.A. Stankovi "RED: A Robust Earliest Deadline Scheduling", 3rd International Workshop on responsive Comuting System, Sept 1993
- [6] C.D. Locke "Best Effort Decision Making for real time scheduling", Ph.D Thesis, Computer Science Departement, Carnegie Mellon University 1986.
- [7] G. Koren, D. Shasha "D over: an optimal On-Line Scheduling Algorithm for Over loaded Real Time System" rapport technique 138, INRIA Fev 92
- [8] Realtime Specification for Java www.rtj.org
- [9] C. L. Liu, J. W. Layland, Scheduling algorithms for multiprogramming in a hard real time environment, Journal of the Association for Computing Machinery, 20(1), Jan. 1973.
- [10] K. W. Tindell, A. Burns, A.J. Wellings, "An extendible Approach For Analysing Fixed Priority Hard Real-Time Tasks", Real-Time Systems 6(2), 1994
- [11] J.P. Lehoczky. "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadline" in IEEE Real-Time System Symposium (1990).
- [12] D. I. Katcher, J. P. Lehoczky, J. K. Strosnider, Scheduling models of dynamic priority schedulers, RR CMUCDS-93-4, Carnegie Mellon University, Pittsburgh, April 1993.
- [13] M.G. Harbour, M.H. Klein, J.P. Lehoczky. "Timing analysis for Fixed Priority scheduling of hard real-time systems Harbour" in IEEE Transaction on Software Engineering, 20(1): 13-28, 1994.
- [14] S. Pertet and P. Narasimhan Proactive Recovery in Distributed CORBA Applications. in IEEE Conference on Dependable Systems and Networks (DSN), Florence, Italy, June 2004.

- [15] K. H. Kim. ROAFTS: A middleware architecture for real-time object oriented adaptive fault tolerance support. In Proceedings of IEEE High Assurance Systems Engineering (HASE) Symposium, pages 5057, 1998.
- [16] H. Kopetz and G. Bauer. "The time-triggered architecture". In proceedings of the IEEE, vol. 91, pp. 112-126, 2003.
- [17] G. Bauer and H. Kopetz. Transparent Redundancy in the Time-Triggered Architecture. In Proceedings of the International Conference on Dependable Systems and Networks (DSN 2000), pages 513, New York, USA, June 2000.
- [18] The RT-SORBET Home Page.
["http://igm.univ-mlv.fr/~midonnet/SORBET.html"](http://igm.univ-mlv.fr/~midonnet/SORBET.html)

A CORBA Bidirectional-Event Service for Video and Multimedia Applications

Felipe Garcia-Sanchez, Antonio-Javier Garcia-Sanchez, P. Pavon-Mariño,
and J. Garcia-Haro

Department of Information Technologies and Communications,
Polytechnic University of Cartagena, Campus Muralla de Mar s/n,
30202 Cartagena, Spain
{felipe.garcia, antoniojavier.garcia,
pablo.pavon, joang.haro}@upct.es

Abstract. The development of multimedia applications using the CORBA A/V Streaming architecture, suffers from a complex software design. This is not a minor drawback in a middleware architecture, intended to simplify the software development process. One source of complexity is the absence of a flexible signaling mechanism to communicate application-dependent control information. As a consequence, developed applications must design parallel communication processes between end points, which obscures the design. Another shortcoming identified, is the rigid flow establishment process, which does not allow the selection of an asynchronous connection setup. In this paper we present an extension of the A/V Streaming service, which addresses these issues. The service proposed provides access to the applications through an integrated bidirectional event-based signaling mechanism. The A/V Streaming extension offers this functionality by means of a CORBA Bidirectional Event Service, also presented in this paper. The A/V Streaming extension under consideration is implemented and comparatively evaluated with the original service, in the CORBA ACE/TAO distribution. Benchmark results validate our proposal, and encourage its practical utilization.

1 Introduction

The design of distributed multimedia applications in the CORBA middleware, adds specific requirements, with respect to conventional distributed applications. This is motivated to achieve an efficient mechanism to combine transmission of multimedia flows between remote objects with the associated control information and signaling. To address this particular scenario, the OMG (Object Management Group), proposed in 2000 the CORBA Audio/Video (A/V) Streaming service specification [1]. The description of the main components that form CORBA's A/V Streaming Service is as follows (Fig. 1):

- A *Stream Interface Control Object*, providing an IDL (Interface Description Language) for controlling and managing streams. This type of information will be handled by the CORBA's ORB (Object Request Broker) using IIOP heading on transport level.

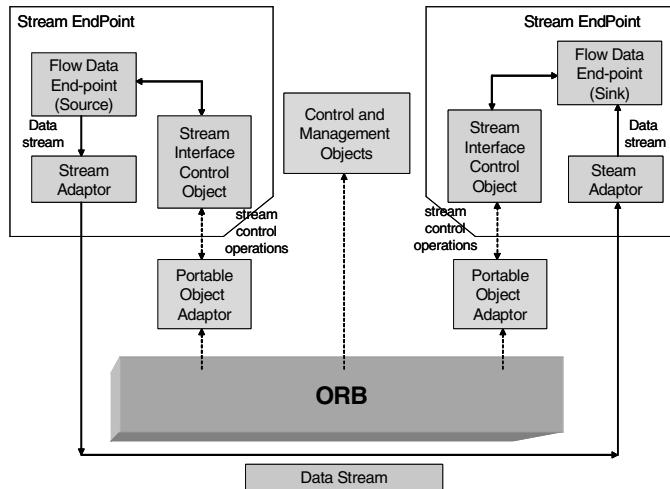


Fig. 1. CORBA's A/V Streaming Service

- *Flow Data Endpoints*, which are either data sources or data sinks [2].
- A *Stream Adaptor* that receives or transmits data frames over a network. The protocols commonly used are UDP, TCP or RTP [2].

The CORBA A/V Streaming Service manages in a different manner two types of communications between distributed objects:

- **Control information** (i.e. establishment signaling flow). This control information is managed by means of conventional CORBA interactions between the flow end points. Therefore, it involves IIOP (Internet Inter-ORB Protocol) messages processed by the CORBA middleware.
- **Data flow**. Once video and/or audio flow parameters are defined, the data flow is established between the signaled end points, by means of the conventional multimedia transport protocol selected (i.e. RTP). Therefore, data flow packets are directly processed by specific (and efficient) multimedia transport protocols. In other words, they are not transported on top of IIOP messages, and are not processed by the CORBA middleware.

Separation of control and flow information in the A/V Streaming architecture allows for a high processing efficiency. Previous evaluation results situate CORBA A/V Streaming as a high-efficiency framework to develop multimedia applications [3] [4][5], offering similar or better performance results than other *middleware* platforms like .Net Remoting and Java RMI. Unfortunately, often the development of large multimedia applications in a CORBA A/V Streaming framework becomes too complex. This is not a minor drawback for a middleware architecture, since its major design objective should be to ease the application development process. We have identified two sources for this implementation complexity increase:

- (1) The signaling mechanism included in the A/V Streaming specification is designed to manage the A/V flow for the establishment, maintenance and termination processes. A fixed set of flow parameters can be configured by means of this signaling procedure. Nevertheless, it does not allow the end-points to communicate any generic application-dependent control information that conventional multimedia applications normally share. For instance, video flow parameters not contemplated in the normal signaling (i.e. frames-per-second or quality thresholds in audio/video reception), speaker identification or textual information (i.e. in a videoconferencing application), etc. Clearly, the objectives, format and type of this control information are heterogeneous. The strict A/V Streaming signaling mechanism is not designed to allow the interchange of this type of information. Consequently, multimedia applications developed must design parallel communication processes between end points, which involves a more complicated and inefficient software design.
- (2) The flow establishment process between a flow source and a flow sink strictly requires the flow source to be active before flow sink is initialized. General multimedia applications would benefit from a connection/disconnection mechanism more open and flexible, which allows the selection of an *asynchronous* establishment where sink and source end points could initiate and wait for the completion of the connection setup.

In many cases, the complexity associated to the development of multimedia applications on top of the CORBA A/V Stream specification, has motivated the software developers to use other type of middleware or application-level tools. In other situations, CORBA multimedia applications have been implemented, employing external signaling systems and protocols to communicate separated control information. This is the case of [6] and [7], where the Session Initialization Protocol (SIP) is used in combination with the CORBA signaling mechanism for the development of a VoIP platform. It can be argued that these examples of use of an Application Level Service instead of a Middleware Layer Component, raise a concern on current CORBA and A/V Streaming service middleware. In author's opinion, the use of CORBA must be rationalized and simplified to give the software designers the maximum of services with low cost of development.

In this paper, we propose an extension of the CORBA A/V Streaming specification, to achieve this make-it-simple philosophy. The objective of our work is to provide an easier platform for the development of multimedia applications. This implies the integration inside the A/V Streaming service of a flexible and powerful mechanism for the interchange of flow-related application-dependent data, which also helps us to redesign the flow establishment process. We base our proposal on the Event Service (ES). The introduction of the Event Service in multimedia applications has been already presented in [9], where it is employed to implement an external communication mechanism that allows for synchronous message passing, and simple direction events. However, the work we present extends and modifies previous work in the following terms:

- 1- The A/V Streaming service is extended, and integrates the implemented application-dependent communication mechanism *inside* the service. This is performed maintaining backward compatibility with existing multimedia applications.

- 2- The flow establishment process is complemented, allowing for the definition of asynchronous establishments, where source or sink end-points can wait for the other end-point to become active.
- 3- The A/V Streaming Service designed, becomes a client of the CORBA Event Service to achieve the aforementioned functionality. This is illustrated in figure 1. However, the current CORBA ES involves an undesired drawback, as the events can only be propagated in one direction. Implementation of a bidirectional transmission of events by means of two unidirectional event connections, implies one more programming burden (for instance, to control the consistent establishment and disconnection of both connections). A relevant contribution of this paper is the proposal, implementation and testing of a modification of the conventional Event Service, which allows a bidirectional propagation of events between end-points. We denote this modified event service as Bidirectional Event Service (B-ES). Of course, the B-ES service could be also accessed by external applications or other software components, as shown in figure 2.

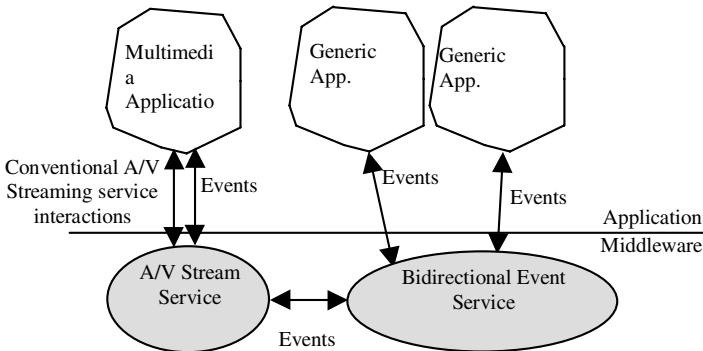


Fig. 2. Interaction between A/V Stream Service and B-ES

The rest of the paper is organized as follows. Section 2 briefly introduces the current Event Service specification. Sections 3 and 4 detail the extension proposed for the A/V Streaming. The section 5 includes the evaluation results obtained. Finally, section 6 concludes.

2 CORBA Event Service Overview

The CORBA Event Service (ES) [6][8] manages the synchronous and asynchronous propagation of messages (called events) among distributed objects. It is based on the traditional policy of Publish/Subscribe, where an edge publishes a particular event and the other end receives it. Therefore, event suppliers and consumers are decoupled: (1) There can be multiple consumers and multiple suppliers of events. (2) Suppliers can generate events without knowing the identities of the consumers. (3) Conversely, consumers can receive events without knowing the identities of the suppliers.

The Event Service design is scalable and suitable for distributed environments. There is no requirement for a centralized server or dependency on any global service. In addition, the event service does not impose higher level policies (e.g. specific event types) allowing great flexibility on how it is used in a given application environment. Furthermore, using the appropriate implementation, reliable event delivery can be supported.

The Event Service is typically used to provide "change notification". When an object is changed (its state is modified), an event can be generated and propagated to all interested parties. For example, when a spreadsheet cell object is modified, all compound documents which contain a reference (link) to that cell can be notified (so the document can redisplay the referenced cell, or recalculate values that depend on the cell). This paradigm is widely applied to different environments, like the automotive industry, medical, general broker networks, P2P networks, etc. The extension of CORBA to mobile environments is based on the Event Service functionality [10].

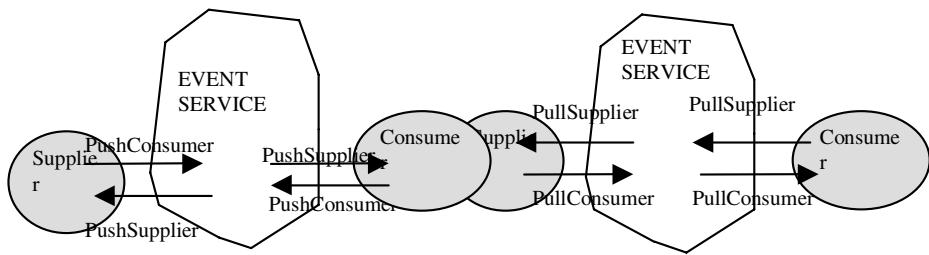


Fig. 3. CORBA Event Service models

Efficient implementations of the CORBA event service already exist and are described in detail in [8]. The main research effort was directed to the design of an efficient event service designed for a real-time environment [11]. Other interesting CORBA based event architecture known as COBEA has been implemented at Cambridge University [12]. This system implements efficient server-side event filtering, the notion of composite events as well as basic access control checks. However, it does not address issues of reliable delivery, events based on multimedia content and the integration of multimedia delivery with the CORBA standard event service. Implementations of the related OMG Notification Service [13] must also address related issues such as filtering, reliable delivery, quality of service and security. This service is an extension to the event service and will be most useful in application scenarios where real time "*filterable*" events are required.

3 A/V Stream Extension: The Bidirectional-Event Service

The Bidirectional-Event Service (B-ES) is our tool developed to provide the desired operation. The traditional implementation of the CORBA Event Service offers different limitations to complement the A/V Video Streaming:

- The ES is based on a Publish/Subscribe policy, opposite to the A/V Stream symmetry of two *EndPoints*. The B-ES implements the same interface for both edges (denominated hybrid) inspired in the A/V stream specification.
- It does not support the current connection establishment of A/V flows.
- Excessive complexity for real time operation.

The new interfaces are primarily based on the IDL (Interface Description Language) interfaces defined in the standard CORBA Event Service [6], with different interesting modifications for the multimedia service. In most cases, we choose to simply expand the functionality provided by the existing standard event service interfaces, rather than to extend these interfaces with new methods in derived interfaces.

The components that perform the CORBA B-ES oriented to Audio/Video Streaming follow the next concerns:

- To respect the symmetry of the A/V Streaming Service.
- To facilitate the integration in a network based on the Event Service.
- To preserve the point-to-point communications and propagations of events.
- To provide the framework to operate with events and multimedia systems for services as videoconferencing, tele-education and tele-medicine systems, etc.

In order to simplify the model, solely Push Events are supported. The Pull model in a bidirectional service may be considered as a push event in the opposite direction, requiring an event from the first *EndPoint*. This chapter defines the interface of the main components of the service.

3.1 StreamChannel (Channel Manager)

The *StreamChannel* is the tool to develop the framework that generates and manages the events produced by the *EndPoints*. It is based on the *ChannelManager* created by the ACE/TAO implementation [14]. According to this implementation, the *StreamChannel* would create a simple channel for the events propagation. This *EventChannel* is not enough to perform the desired bidirectionality and management of the A/V streaming events. In the next section, the process implementing the event propagation in the two ways is explained, based on the creation of two *EventChannel*, each one for a direction of the communication.

Interestingly, it should be emphasized that the *StreamChannel* presented might be used as a typical *EventChannel*, when several definitions are unused or set to the default values. It is possible to facilitate events in just one direction for unspecified applications where a simple events channel is required. The prototype of the interface is as follows:

```
interface StreamChannel
{
    typedef sequence<string> stringseq;
    EventChannel creates(in string channel_id, in ChannelType type,
        in DispatcherType DisType, in Properties) raises (unable);
    void destroy (in string channel_id);
    ChannelType knowchannel(in string channel_id);
```

```

void handle_events();
void register_handler(in Handler HandlerName);
void remove_handler(in Handler HandleName);
};

```

Summarizing the basic work, this interface introduces the channel for event propagation. The method *creates* implements the creation of the channel for events. *Destroy* removes a channel previously created.

The *Dispatcher Type* variable is an insight difference with the traditional Event Service. The Dispatcher operation (event resolution and transmission) of the specification does not permit the desired *StreamChannel* job. Therefore, to enhance the functionality of the traditional ES, a second dispatcher is created, namely the *StreamDispatcher*. It carries out the event transmission for a *StreamChannel*, where the two *Endpoints* are identical.

Another relevant task is done by the method *knowchannel*. It is necessary to know the type of channel that is employed. For instance, it is not the same a *StreamChannel* that a common *EventChannel*. In principle, a *StreamChannel* allows the propagation of events of an *EventChannel*, but normally, a *StreamChannel* is used for A/V Streams events, and the *EventChannel* is for other cases. Therefore, a method to distinguish both types is necessary.

The methods *handle_events*, *register_handler* and *remove_handler* are the same that the ones in the definition of the Event Channel. They are used to define events, to manage them and to remove them.

The interface might be completed for security, multicast, etc. operations of the B-ES, enhancing its behaviour and increasing the complexity of the Event Service.

3.2 Event Channel

The *EventChannel* implements the *StreamChannel* referred in the previous point. Notice that the *EventChannel* is just an extension of a traditional *EventChannel* but adding the *HybridAdmin* method. The interface follows the next template:

```

interface EventChannel
{
    attribute ChannelType type;
    attribute Properties props;
    CosumerAdmin for_consumers();
    SupplierAdmin for_supplier();
    HybridAdmin for_hybrid();
    void destroy();};

```

Basically, it is the same *EventChannel* mentioned above by the current specification, adding the method to create the *HybridAdmin* functionality necessary for the A/V Stream event transmission. Its interface is detailed below. Moreover, the attribute *ChannelType* makes the difference between the *StreamChannel* and the *EventChannel*.

The name of this “prototype” is not changed from the original specification to facilitate the integration with previous implementations of the *EventChannel*. We should remark that the name *StreamChannel* is used for the Channel Manager and for a type of the *EventChannel*.

3.3 Hybrid Administrator

The Hybrid Administrator is the service administrator of the conventional component of connection of the *StreamEndPoints* to the service. In this enhanced component the function *create_flow* allows to the Event System to create the flow channel. In this case, the creation of the flow channel follows the specification of the A/V Stream Service, using an independent channel where the Event Service has not any action. Therefore, its task has two main purposes:

- To know the *Flow Data StreamEndPoint* location. Basically, the addresses and ports, but we include additional information as the type of communication, image size, etc.
- To order the connection to both *StreamEndPoint*, informing which one executes the active connection (normally the first connected one).
- In the case of systems as videoconferencing, where two flows are created the information is doubled: two ports, two image sizes, etc.

```
interface HybridAdmin
{
    void create_flow (in string flowName, in string format,
Hybrid peerName1, Hybrid peerName2) ;
    void destroy (in string flowName) ;
    void peer_stream (Hybrid peerName) raises (noAvailable,
noSupported) ;
};
```

The method *destroy* completes the functionality of the *HybridAdmin*, where a flow can be destroyed. *Peer_Stream* allows to the Event Service to know exactly the location and information of the different *EndPoints*. These methods must be utilized twice, one for each *EndPoint*. The *noAvailble* or *noSupported* exceptions concern about the failure of the process when the peer is not present or is not able to support the flow required.

Notice that *create_flow* and *destroy* do not appear in the Event Service specification. They do not support the event transmission (included in the *EventChannel*). They are methods to create the A/V flow, independently of the events processing. This idea is based again on the A/V Stream Service specification. Both methods are included here, generalizing the use of this *HybridAdmin* not only for the *StreamChannel*, even they are useful to set the establishment of the A/V Stream Service directly from two *EndPoints*. Indeed, a model of events (additional to the classical push and pull models) is presented. It is no implemented a “Pushpeer_stream” or a “Pull-peer_stream” as the traditional service. This further helps to simplify the system.

3.4 ProxyHybrid Interface

The *ProxyHybrid* Interface extends a particular version of the Supplier/Consumer structure. The interface shows a special prototype where the functions of the conventional supplier and consumer are mixed. In fact, the hybrid model has the Supplier and Consumer roles, and behaves as one or another depending on the situation:

```

interface ProxyHybrid:Hybrid
{
void connector (in string channel_id);
void acceptor(in string channel_id);
void peer_acceptor();
void accept(in Hybrid peerHybrid, in string flowName);
void connect(in Hybrid peerHybrid, in string flowName);
void get_flow(in string flowName ) raises (noAvailable);
void complete();
void handle_events();
};

```

The methods *connector* and *acceptor* are inherited from the traditional system of a *ProxyPushConsumer* and *ProxyPushSupplier* respectively. In the *ProxyHybrid*, they have a different mission. The *connector* is a factory of functions in charge of the events produced by the other *StreamEndPoint*. The *Acceptor* creates the interface to support the events generated by its own *StreamEndPoint*.

The methods *peer_acceptor*, *accept* and *connect* follow the connection procedure of the *Hybrid EndPoint* to the Event Service. Therefore, the method *accept* allows the connection from an *EndPoint*. The method *connect* consists of the connection from the ES to an end service, and the *peer_acceptor* is in charge of the connection from the remote *EndPoint*. The method *getflow* is utilized to add an *EndPoint* to a channel. In the case that the *StreamChannel* is operational, the result is affirmative. Otherwise, it will receive an error indication or will remain waiting.

Handle_events is the method to transmit the events, and to finish (*complete*) when an event is received and finished. It should be noted that the method *complete* was also used to accept the A/V flow connection.

4 Implementation Details

The implementation of the enhanced A/V Stream Service through the B-ES is based on two concepts:

- The symmetry of the A/V Streaming model.
- The traditional Event Service connection calls.

Although both services are asymmetric, their integration makes possible the desired A/V Stream Service with event propagation. This design simplifies and supports the bidirectional exchange of A/V flows and event. It is accomplished over a Linux System (Red Hat v. 8.2), and written in C++ language. The CORBA distribution selected is the ACE/TAO, due to its appropriate documentation, its own A/V Stream Service and Event Service implementations and because it is a free distribution framework.

The Top Level Design is conditioned by the *Hybrid* interface that offers the maximum functionality. Figure 4 shows the component relationship of the Hybrid and the B-Event Service according to the Unified Modeling Language (UML) format.

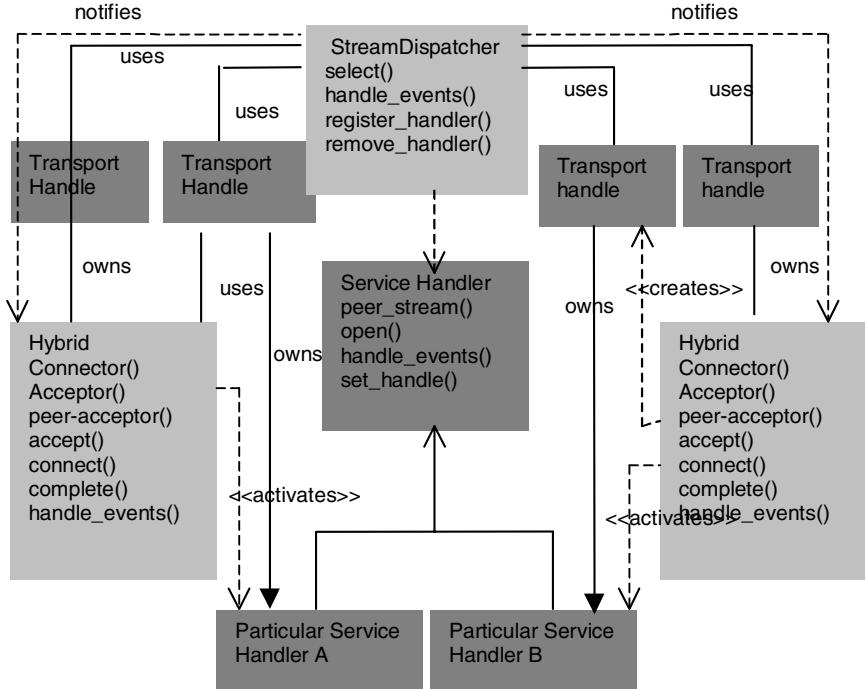


Fig. 4. Component relationship

Unfortunately, there are several methods (public and private) that do not have an IDL representation. For this reason, the UML representation does not offer a straightforward comprehension of the complete processes of connection and event propagation. They will be explained and detailed in the next subsections.

The Bidirectional Event Service (B-Event Service) treats to solve the problem of multiple *EventChannels* connecting a large number of *StreamEndPoints* by means of different threads. An *ORBThread* [4] is listening to incoming connection request.

Based on the classical Event Service, the *StreamChannel* (*ChannelManager*) is a factory of *ChannelEvents* of type *StreamChannel*. Every time, a new channel is required, the *StreamChannel* configures the appropriated resources, event types, etc., and starts a new thread for each *StreamChannel*. Simultaneously, a second channel of *StreamChannel* might be created for the events in the opposite direction. The decision of one (unidirectional) or two (bidirectional) channels is taking depending on the type of application. In subsequent implementations, the user will be able to choose the number of flows and their directionality.

4.1 A/V StreamEndPoint Connections

The first operation is the connection of the *StreamEndpoints*. Moreover, it is necessary to consider the connection and initialization of the A/V Flow Channels (not connected through the B-Event Service).

The *StreamEndpoints* must require the connection to the B-Event Service through a *request* message, carrying out the *StreamEndPoint* (Name) and type of A/V Stream (Video Type, format, Transport Protocol, etc.) identifiers. All these features conform a particular *connect_packet*. This *connect_packet* is sent by the *StreamEndPoint* to the B-Event Service. The *StreamEndPoint* creates a *Hybrid* thread that supports the connection to the *Event Service* and the subsequent event propagation (Fig. 5).

When the B-Event Service (B-ES) receives the request of a *StreamEndPoint* (by issuing a *connect_packet*), it registers all the information conforming a database with

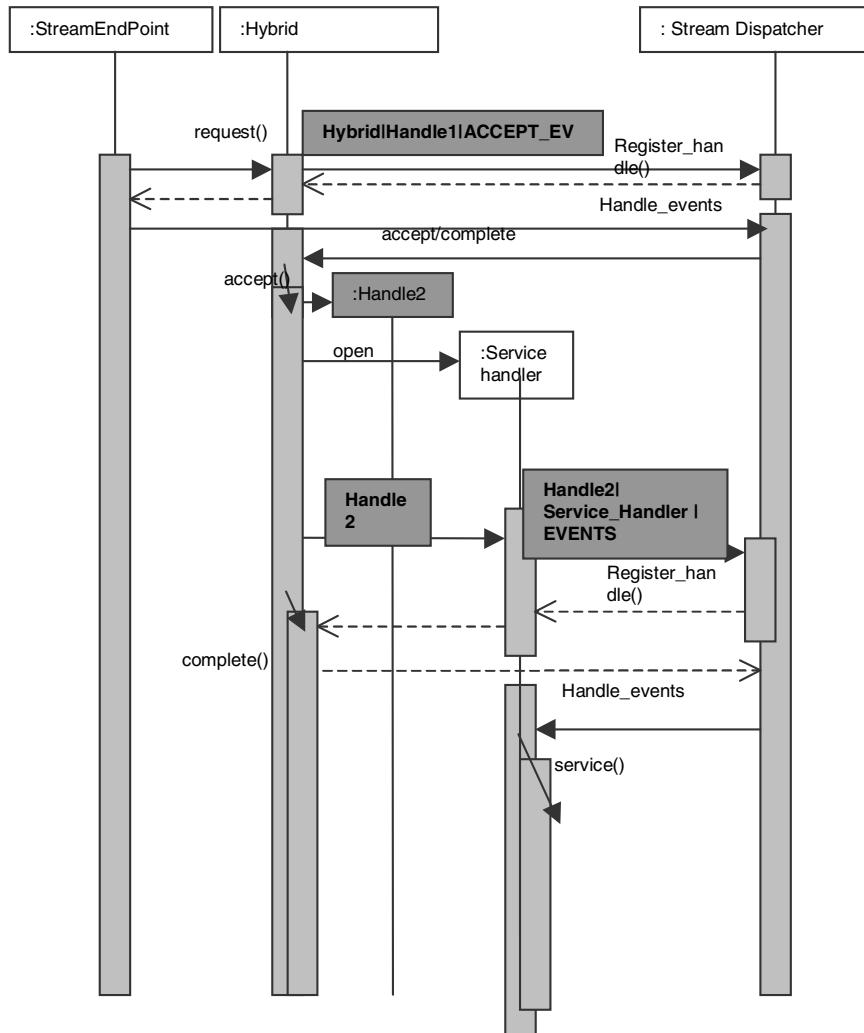


Fig. 5. Dynamics of the StreamEndPoint-Event Service

EndPoints and *requests*. The decision of connection is taken by the *StreamDispatcher*, when a *connect_packet* of a *StreamEndPoint* arrives. By default, the current implementation does not offer the possibility to select between the traditional Dispatcher and the *StreamDispatcher*. It may be added in later implementations.

The information of the *connect_packet* is saved by the B-Event Service in the database and keeps waiting for the connection of the second *StreamEndPoint*. It is also possible to establish the connection of the second *StreamEndPoint* from the B-Event Service. In this case, the information of the Name of the second *StreamEndPoint* might be inscribed in a *NameService*. The B-Event Service obtains the reference of that *EndPoint* and sets up the connection. In this situation the *connect_packet* does not offer the same information (no Video Type, format, Transport Protocol, etc. are provided). Now, it is just a packet informing that an Event Service requires its connection. The *StreamEndPoint* is able to reject the request or to establish the connection. In the later case, the connection is established as commented before, but without information of the *StreamEndPoint* destination. At the end of this process, both *EndPoints* are connected.

Next step is to create the A/V flow and the *StreamChannel*. The first task is done by *StreamChannel* (channel for events). The channel for events is programmed as an *EventChannel* of type *StreamChannel*. The *StreamChannel* is invoked to create these type of event channels.

Depending on the service *request*, one or two *StreamChannels* are created (i. e. videoconferencing requires two channels). First, a channel for a direction of the communication is created. Then, the *ChannelStream* does an *accept* call to the *hybrid* of one *EndPoint*, and execute *complete* to the *hybrid* of the other *EndPoint*. The way of the events is from the first *StreamEndPoint* to the second. For the second direction, the process is similar but changing the roles.

When both *StreamChannels* are created, the creation of the A/V flow channels is performed as events inside the channel of events. One requirement is when two directional flows are requested. It is necessary to previously establish both *StreamChannels*. If events are used in just one way, the second *StreamChannel* might be destroyed.

The *create_flows* method sends to each end the corresponding call of connection, as if the B-ES was a *StreamEndPoint*, but changing its own reference by the reference of the other *StreamEndPoint*. This information is encapsulated as an event of “request” that the *StreamEndPoint* interprets as a *request* connection of an A/V Stream. Therefore, both edges execute the traditional connection according to the obtained reference. In this process the B-ES does not have any type of participation.

This idea might be extended to diverse A/V flow channels from the same *StreamEndPoint* origin, when the service requires it (i.e. for video-on-demand one flow for the sound and other for the image are needed).

An important feature is that connection is completely asynchronous. Each *StreamEndPoint* connects when it wants to access the service. It is possible thanks to the additional *Hybrid* interface that implements the connection, service and events.

4.2 A/V Stream Events Management with the B-ES

In the *hybrid* interface of the *StreamEndPoint*, the event management is performed at the edges. The *StreamEndpoints* invoke the B-Event Service interface each time they

sent an event. The events encapsulate the data associate to a particular event. The event of stream connection is just the necessary once to create the A/V flows. The number and complexity of the events depend on the particular application. Two conditions of design are imposed:

- The same thread generated by the *hybrid* interface takes care of local and remote events.
- In case of event collisions, the remote event is served previously.

The dynamic work of the B-Event Service is basically the same than the one of a traditional Event Service when the *Supplier* and *Consumer* are connected and an event occurs. The service of events is asynchronous. Figure 6 illustrates the asynchronous operation. It starts when the event arrives to the *ProxyHybrid* connected to the origin *EndPoint*. It invokes an asynchronous operation to read the event for the input event queue. Events usually arrive multiplexed with other events. The *ProxyHybrid* executes methods to resolve it. When the event is demultiplexed, the *ProxyHybrid* gets from the system (*StreamDispatcher*) the handler (permission) to execute the method *handle_event*. This *handle_event* propagates the events around the *Hybrids* connected to the *StreamChannel*.

Finally, the event arrives to the desired *StreamEndPoint*. Each *ProxyHybrid* is able to deliver directly the events to the *Hybrids* connected to the *StreamChannel*. The reason is that *ProxyHybrid* has all the methods to send and receive events. The only requirements are to know the *EventChannel* and the specification of the events.

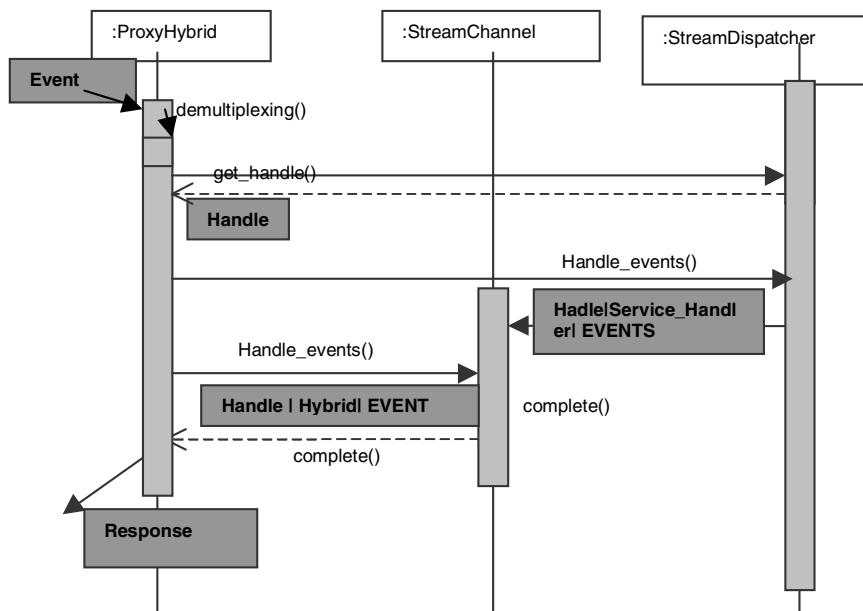


Fig. 6. Event propagation

The supported events are typed by default in the Event Service of ACE/TAO. The programmer may implement all the particular events.

Notice that here, the concepts of *Supplier* and *Consumer* do not take place. Although, a *StreamEndPoint* sends the event and a second *StreamEndPoint* receives it, they are *Hybrids* changing appropriately its role.

5 Performance Evaluation

In this section, we present a set of evaluation results for the Bidirectional Event Service and the A/V Streaming extension implemented. The results have been obtained from a benchmark architecture in a commercial computer Pentium IV 2 GHz, 512 Mbytes RAM, 80GB hard-disk. The A/V Streaming and the Event Service have been implemented using the CORBA ACE/TAO distribution [15] [16].

In a first stage the operation of the B-ES is evaluated in terms of the time required for the establishment of a large number of bidirectional event connections between end points. Both the Event Service and the set of end points are initiated in the benchmark machine. The results are compared to the time required to complete (1) unidirectional consumer connections, (2) unidirectional supplier connections, and (3) the sum of consumer and supplier connections. Results displayed in figure 7 show that the computational cost of the bidirectional connection is closer to the cost of the consumer unidirectional connection in the original Event Service, and below the sum of the consumer plus supplier connections.

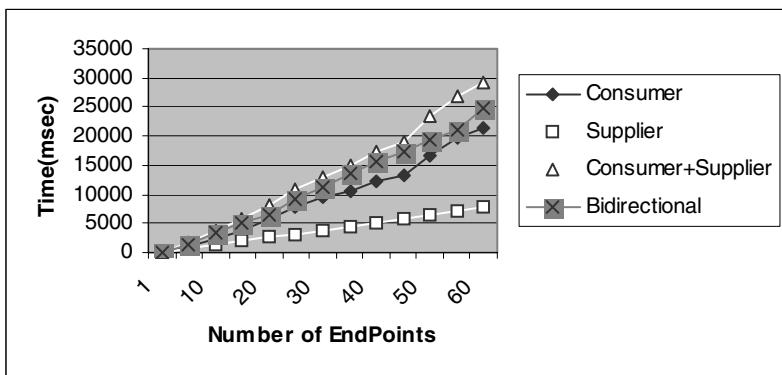


Fig. 7. *StreamEndPoint* Connection

In a second evaluation stage, we are interested in the computational efficiency of the establishment of the A/V flow. Note that, in the new A/V service presented, the A/V flow establishment also involves the setup of a bidirectional event channel between end points. To make a fair comparison, we contrast two situations:

- (a) The connection setup of an A/V flow with the new service presented, which includes the establishment of a bidirectional event connection.

- (b) The connection setup of an A/V flow with the traditional A/V service together with the establishment of two event connections between end points, a consumer and a supplier one.

Figure 8 depicts the required time for the establishment of a given amount of A/V connections, in the two aforementioned situations (a) and (b). Results reveals that the computational cost of the process is smaller for the new A/V Streaming service presented.

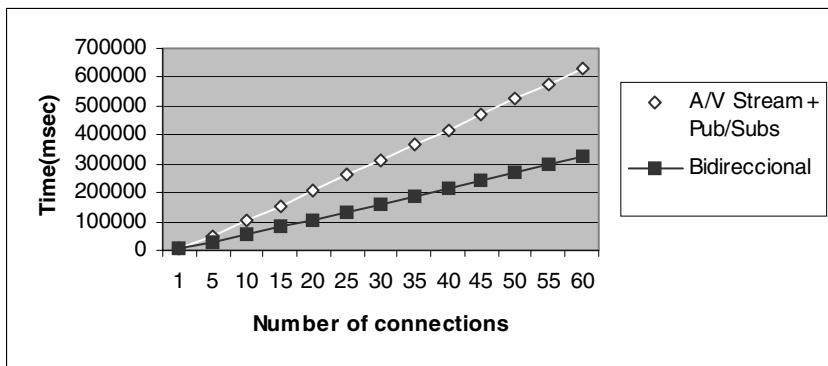


Fig. 8. A/V Flow Channel Establishment

Figure 9 comparatively evaluates the event propagation process between end points, also for the cases (a) and (b). The statistic selected is the event propagation delay: the elapsed time from the event creation in the supplier *EndPoint*, to the event consumption in the sink *EndPoint*. Not surprisingly, the results show a slight improvement for the new bidirectional service. The advantage obtained is due to a simpler and lighter software design, where the *Hybrid* or *ProxyHybrid* components directly propagate events without further processing operations.

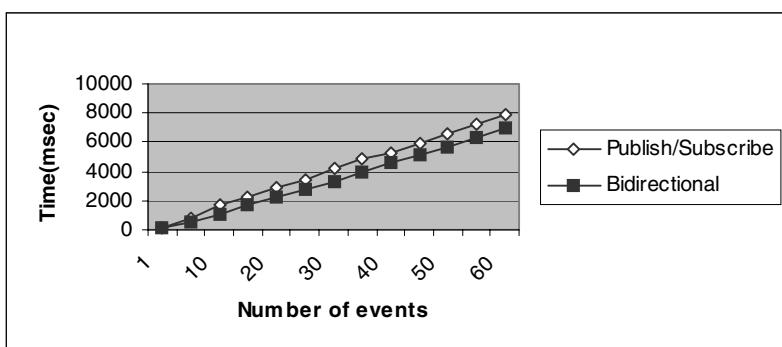


Fig. 9. Event Propagation analysis

In summary, the results obtained validate the new service proposed, and exhibit a better or similar performance than the combination of the conventional A/V Streaming service and the traditional Event Service.

6 Conclusions

This paper describes an extension of the CORBA A/V Stream Service by modifying the Event Service. The objectives of our work are the integration in the A/V Streaming service of a set of functionalities, identified as major demands from multimedia application developers: (1) A more powerful mechanism to communicate control information between flow *EndPoints*, (2) a more flexible flow establishment procedure.

To accomplish this task, the A/V Streaming service proposed interacts with a modified Event Service, also presented in this paper: the Bidirectional Event Service (B-ES). The B-ES complements the traditional Event Service with the capability of establishing bidirectional connections of events.

The proposed A/V Streaming and Event Service have been implemented by using the CORBA ACE/TAO distribution [13]. The model has been tested and evaluated, showing similar or better performance results than the conventional A/V Streaming Service and the traditional Event Service.

Acknowledgement

This research has been funded by the Spanish MCyT grant TEC2004-05622-C04-02/TCM(ARPaq).

References

1. "Audio/Video Stream Specification", Object Management Group, January 2000.
2. Munjee S, Surendran N, Krishnamurthy Y, Schmidt DC, "The design and performance of a CORBA Audio/Video Streaming Service", in IEEE Proceedings of the Hawaiian International Conference on System Science, Hawaii (EEUU), pp. 8043-8059, Jan. 2001.
3. D. L. Levine, S. Flores-Gaitan & D. C. Schmidt "An Empirical Evaluation of OS EndSystem Support for Real-Time CORBA Object Request Broker". Multimedia Computing and Networking 2000 (MMCN00). San Jose, California, 25-27 January 2000.
4. F. Garcia-Sanchez, A.J. Garcia Sanchez, J. Garcia-Haro, "Performance Evaluation of Video Flows Integration over IP Networks using TAO" in Proceedings of the 9th IFIP/IEEE Symposium of the Integrated Networks Management. Nice (France), May 2005.
5. F. Garcia-Sanchez, A.J. Garcia Sanchez, J. Garcia-Haro, "Performance Evaluation and Implementation Details for the CORBA A/V Stream Service for Video Communications", in Proceedings of the 23rd IASTED Conference on Parallel and Distributed Computing and Networks, Innsbruck (Austria), pp. 436-443, February 2005.
6. Kundan Singh, Gautam Nair and Henning Schulzrinne "Optimization of Signaling Traffic in Centralized Conference using SIP", in Proceeding of the WSEAS ICOMIV 2002, Skiatos (Greece), pp 2931- 2936, 2002.

7. RV Prasad, R Hurni, HS Jamadagni "A Scalable Distributed VoIP Conferencing using SIP" in Proceedings of the 8th IEEE Symposium on Computers and Communications," Antalya, Turkey, pp. 608-613, July 2003.
8. "Event Service Specification", Object Management Group, March 2001.
9. Edouard Lamboray, A. Zollinger, O. Staadt, M. Gross, "Interactive multimedia streams in distributed application", Computer & Graphics 27, pp. 735-745, 2003.
10. M. Caporuscio, A. Carzaniga, A. L. Wolf, "Design and Evaluation of a Support Service Publish/Subscribe Applications", IEEE Transactions on Software Engineering, vol . 29, no 12, December 2003.
11. D. Gill, F. Kuhns, D. Levine, D. C. Schmidt, B. S. Doerr, R. d E. Schantz, and A. K. Atlas, "Applying Adaptive Real-time Middleware to Address Grand Challenges of COTS-based Mission-Critical Real-Time Systems", in Proceedings of the 1st International Workshop on Real-Time Mission-Critical Systems: Grand Challenge Problems, IEEE, Phoenix, Arizona, November 30, 1999.
12. C. Ma and J. Bacon, "COBEA: A CORBA Based Event Architecture" in Proceedings Of the COOTS'98 Conference Santa Fe (New Mexico), pp. 117-131, April 1998.
13. OMG. Notification Service. RFP. OMG Document No. Telecom/96-11-03. Nov. 2003.
14. Desmons Chambers, Gerard Lyons and Jim Duggan, " Stream Enhancements for the CORBA Event Service", in Proceedings Of the 9th ACM Multimedia Conference, Ottawa, Ontario, Canada, pp. 61-69, October 2001.
15. A. Gokhale and D. Schmidt. "Measuring and Optimizing CORBA Latency and Scalability over High-Speed Networks", in Transactions on Computing, vol. 47, no. 4, 1998.
16. Douglas C. Schmidt, Computer and Science Engineering of Washington University: "<http://www.cs.wustl.edu/~schmidt/TAO.html>".

GREEN: A Configurable and Re-configurable Publish-Subscribe Middleware for Pervasive Computing

Thirunavukkarasu Sivaharan, Gordon Blair, and Geoff Coulson

Computing Department, Lancaster University, Lancaster, LA1 4YR, UK
{t.sivaharan, gordon, geoff}@comp.lancs.ac.uk}

Abstract. In this paper we present GREEN a highly configurable and re-configurable publish-subscribe middleware to support pervasive computing applications. Such applications must embrace both heterogeneous networks and heterogeneous devices: from embedded devices in wireless ad-hoc networks to high-power computers in the Internet. Publish-subscribe is a paradigm well suited to applications in this domain. However, present-day publish-subscribe middleware does not adequately address the configurability and re-configurability requirements of such heterogeneous and changing environments. As one prime example, current platforms can-not be configured to operate in diverse network types (e.g. infrastructure based fixed networks and mobile ad-hoc networks). Hence, we present the design and implementation of GREEN (Generic & Re-configurable EvEnt Notification service), a next generation publish-subscribe middleware that addresses this particular deficiency. We demonstrate the configurability and re-configurability of GREEN through a worked example: consisting of a vehicular ad-hoc network for safe driving coupled with a fixed wide area network for vehicular traffic monitoring. Finally, we evaluate the performance of this highly dynamic middleware under different environmental conditions.

1 Introduction

Recent advance in wireless network technologies (e.g IEEE 802.11) and computational devices (e.g. PDA, PC) have created opportunities for the vision of pervasive computing applications [1], which embrace both fixed infrastructure based (wired and wireless) networks and wireless ad-hoc networks. Event based communication based upon the publish-subscribe model is well-suited to pervasive computing applications, as it presents an asynchronous and decoupled communication model [2], [3],[4],[44]. Notably, pervasive computing applications operate across highly heterogeneous environments in terms of network types (e.g. WAN, MANET) and device types. However, many publish-subscribe middleware have specifically targeted fixed infrastructure based networks e.g SIENA [5], Gryphon [6], Hermes [7] and JEDI [8]. At the other extreme STEAM [9] is specifically designed for wireless ad-hoc networks. We argue that publish-subscribe middleware that operates over a single homogenous network environment (i.e. WAN or MANET) and offers a single (or fixed) interaction type (i.e topic based or content based) cannot cope with the diversity of environmental constraints and requirements presented by pervasive computing applications. Dealing with such extreme heterogeneity is a fundamental

challenge for future publish-subscribe middleware and one that is demonstrably not addressed by existing platforms. To overcome this problem we believe it is necessary to build highly configurable and dynamically reconfigurable publish-subscribe middleware which can be deployed in heterogeneous network types and heterogeneous device types and meet application and environment specific requirements.

This paper presents GREEN, a deployment and run-time reconfigurable publish-subscribe middleware. GREEN follows the well established approach to the development of reflective middleware [10], [11]; it uses the marriage of OpenCOM components [12],[13], reflection [14] and component frameworks (CFs) [15] to yield a configurable, reconfigurable and evolvable publish-subscribe middleware architecture. In particular, GREEN is configurable to operate over heterogeneous network types (e.g. MANET and WAN) and supports pluggable publish-subscribe interaction types (i.e. topic based, content based, context, composite events). Further, the underlying event routing mechanisms are reconfigurable to support selected interaction type in different network types. The distributed event routing and event filtering is underpinned by pluggable distributed event broker overlays; we create overlays of event brokers to suit contrasting network types.

In the remainder of this paper we first, in section 2, describes our approach to building reconfigurable middleware. Then, in section 3, we present the GREEN architecture. In section 4, we describe the implementations of GREEN configurations based upon a case study and then in section 5, provide performance results of our work to date. Finally we survey related work in section 6, and present our conclusions in section 7.

2 Building Re-configurable Middleware: Lancaster Approach

It is clear that GREEN middleware must accommodate an increasing diverse range of requirements arising from the needs of both applications and underlying systems (e.g device types, network types). Moreover, it is clear that to achieve this accommodation GREEN must be capable of both deployment-time configurability and run-time re-configurability. Unfortunately, the current generation of mainstream middleware is, to a large extent, heavyweight, monolithic and inflexible and, thus, fails to properly address such requirements. It is important to note, the approach for achieving *re-configurability* is important in itself. Therefore, this section describes the approach taken by GREEN to address these requirements. GREEN follows Lancaster's well-founded approach to building re-configurable middleware platforms [10],[11]. GREEN is built using our well founded lightweight component model [12],[13], uses reflective techniques [10] to facilitate re-configuration, and employs the notion of component frameworks (CF) to manage and constrain the scope of reconfiguration operations.

Component technology [15] has emerged as a promising approach to the construction of configurable software systems. With component technology, one can configure and reconfigure systems by adding, removing or replacing their constituent components. Importantly, components are packages in a binary form and can be dynamically deployed within an address space. Additional benefits of component technology include increased reusability, dynamic extensibility, improved understandability and better support for long term system evolution. It should be

noted, however, that current component models (e.g. Enterprise JavaBeans, Microsoft COM) provide little or no support for integrity management; system integrity can be easily compromised if run-time reconfiguration operations are not carried out with great care. In our previous work we have addressed this problem and presented our component model known as OpenCOM [12], [13]. OpenCOM is a lightweight, non-distributed, language independent component model that is independent of any infrastructures, thereby enabling GREEN middleware itself to be built using components. Figure 1 shows the basic elements of the component model. Components interact with other components through interfaces and receptacles. Interfaces are expressed in terms of sets of operation signatures provided by the component. Receptacles are required interfaces that are used to make explicit the dependencies of a component on the other components. Bindings are associations between a single interface and a single receptacle (within an address space).

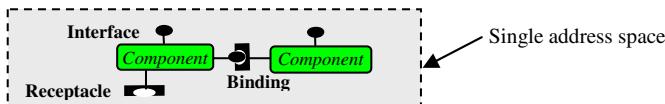


Fig. 1. Basic elements of OpenCOM component model

Furthermore, our component model is highly *reflective* [10]; in other words, the component configurations that comprise the middleware are associated with causally connected data structures (called meta-structures) that represent (or, in reflection terminology, ‘reify’) aspects of the component configurations, and offer *meta-interfaces* through which these reified aspects can be inspected, adapted and extended. The use of reflection facilities the management of run-time reconfiguration of the middleware, and also helps address the issue of integrity management referred above.

The second key underpinning of GREEN is the adoption of the concept of *component frameworks* (CFs) to architect and build GREEN. CF was originally defined by [15] as ‘collections of rules and interfaces that govern the interaction of a set of components plugged into them’. Each CF targets a specific domain and embodies ‘rules, interfaces and components’ that make sense in that domain. The rules define valid configurations (or graph) of components. Crucially CFs actively police attempts to plug-in or swap new components according to these rules. It is important to note, GREEN architecture consists, a set of hierarchically composed component frameworks (more on this later). Furthermore GREEN applies our notion of *deep middleware* [11], [46] in which the middleware platform reaches down into the (heterogeneous) network to provide flexible communications services with which to support a range of publish-subscribe interaction types at the application level.

3 The GREEN Architecture

3.1 Overview

This section describes the GREEN architecture, a generic, configurable, reconfigurable and reflective publish-subscribe middleware to support pervasive computing application development. GREEN uses OpenCOM as its component

technology, is built as a set of component frameworks (CFs), and is based upon the generic middleware framework proposed in [46]. This generic middleware framework developed at Lancaster offers a two layered architecture, where the higher layer is an *interaction framework* that takes plug-in interaction types (e.g publish-subscribe, RPC, tuple-space); the lower layer is an *overlay framework* which takes plug-in overlay implementations (e.g application level multicast overlays).



Fig. 2. The overall architecture

This approach separates middleware interaction types from the underlying overlay network implementations as seen in figure 2, providing configurability, re-configurability and re-use. Importantly, GREEN concentrates on ‘publish-subscribe’ based interaction types and ‘event broker overlays’.

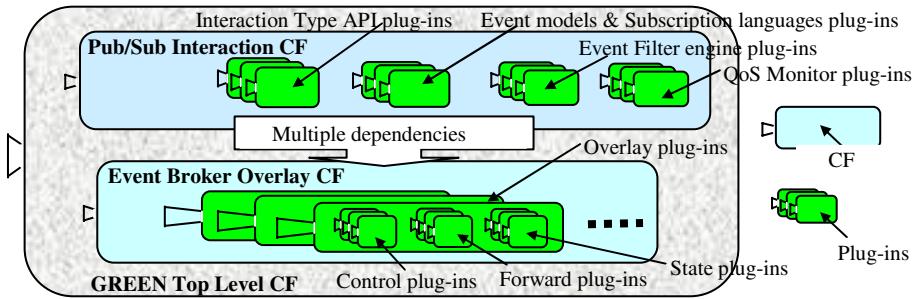


Fig. 3. GREEN Architecture

The GREEN architecture consists of two main component frameworks (CFs); 1) publish-subscribe interaction CF and 2) an event broker overlay CF (see figure 3). The publish-subscribe interaction CF is configured by plugging in different publish-subscribe interaction type implementations e.g. topic based, content based etc. The event broker overlay CF is similarly configured by plugging in different overlay protocol implementations, e.g. probabilistic multicast overlay for ad-hoc networks and the Scribe [19] overlay for WAN etc; where overlay networks are virtual communication structures that are logically laid over an underlying physical network such as Internet or ad-hoc networks [17].

Finally, the GREEN top level CF (see figure 3) which is itself composed of two layers of the above mentioned CFs; mandates the appropriate layer composition between interaction CF and overlay CF and configures the two CFs to provide distinct implementations of GREEN configuration(s). For example, 1) proximity and content based publish-subscribe interaction type underpinned by probabilistic

multicast overlay for ad-hoc networks and 2) a content based publish-subscribe underpinned by Scribe [19] overlay for wide area networks (WAN) etc.

3.2 Publish-Subscribe Interaction Component Framework

The main function of the pub-sub interaction CF is to provide various pub-sub interaction types such as topic based and content based etc. Therefore, over time the framework may be configured as a topic based pub-sub personality where subscribers can make topic based subscriptions, or change to content based pub-sub personality for highly expressive content based subscriptions, or, context based system (e.g. location and proximity based as in [9], [22]). Within the pub-sub interaction CF changes can be made at distinct levels (illustrated in figure 3). Firstly each interaction type API plug-in can be replaced; e.g. topic based interaction type API is replaced by content based interaction type API. This re-configuration is performed according to application requirements. Secondly, different subscription language plug-ins such as FEL (described later) and XPATH [45], event data models (e.g. strings, sequences of values (tuples), name-value pairs, XML based, objects) and the associated event filter engine implementations can be plugged-in. The decision on the subscription language plug-in and the event data model to configure, can be made in light of device context, e.g. resource scarce embedded devices can use simple strings as event data model instead of verbose XML. Furthermore, run-time reconfigurations can be made in light of changes in the quality of service (e.g. throughput). For example, content based interaction personality can be replaced by a topic based interaction to help obtain high event throughput in the system.

In order to test and evaluate the pub-sub interaction CF, we have implemented 1) interaction type API plug-ins (i.e topic based, content based and proximity based), 2) an extensible subscription language known as FEL and its associated event filter engine plug-ins and CLIPS -a composite event specification plug-in and 3) QoS monitor plug-ins (i.e. TCB – discussed later).

<pre> Exp ::= a S b a S b [T] S ::= { Subject names , * } T ::= t t & T t ::= % name % operator \$ value \$ name = { attributes names } operator = { =, >, < } value = { string or numeric values } a = // , b = / , * = Any subject Exp - Filter expression </pre>	<pre> Exp ::= a S b ?#CONTEXT#operator\$value\$ a S b [T]?#CONTEXT#operator\$value\$ S ::= { Subject names , * } T ::= t t & T t ::= % name % operator \$ value \$ name = { attributes names } operator = { =, >, < } value = { string or numeric values } a = // , b = / , * = all subjects , CONTEXT = {DISTANCE,} </pre>
---	--

Fig. 4. a) Grammar of FEL

b) Grammar of extended FEL for Context

Example subscriptions	FEL Filter Type plug-in
1) //stock/[]	Topic
2) //stock/{%name%=\$IBM\$&%exchange%=\$NYSE\$&%price%>\$50\$}	Topic+Content
3) //RoadTraffic/{%type%=\$TrafficLight\$}?#DISTANCE#<\$15\$	Topic+Content+Context

Fig. 5. Example subscription types in FEL

FEL (Filter Expression Language) is an ‘extensible’ language that we have defined and implemented, which enables the definition of topic filters, content filters and context filters depending on the configured FEL plug-in type. The grammar of the subscription language is illustrated in figure 4. A few example subscriptions in FEL are illustrated in figure 5. Notably, example three is a context based subscription, specifically a location context (i.e. proximity) and enables the subscriber (e.g. vehicle) to receive ‘traffic light information generated from traffic lights which are located within 15m distance’. The context filter plug-in transparently handles adding context data (e.g. GPS location coordinates) to the original events published by the application. Importantly, the application programmer need not deal with context data.

The XML based event data model was chosen in the implementations as it’s easily extensible, interoperable and it is platform and programming language independent, however it suffers from high processing overhead. The FEL subscription language plug-in is suited for individual event notifications. However, in some applications subscribers may require to specify interest in the occurrence of multiple related events. Specifying interest in composite events on top of a content based publish-subscribe system is a powerful interaction type for many distributed applications [23]. Therefore, the popular rule-based inference engine CLIPS(C Language Integrated Production System) [26] is provided as a additional plug-in in the framework for applications which require rule-based composite events specification support. When the CLIPS plug-in is configured, a subscriber can submit ‘event-condition-action’ based subscriptions. Some of the benefits of using CLIPS language are its platform and language independence and the high efficiency of the inference engine. The internal implementation of CLIPS is based upon RETE nets [27]. Furthermore, it is feasible for new subscription languages and event data models to be dynamically integrated into the framework at a later date.

3.3 Event Broker Overlay Component Framework

The primary function of the event broker overlay framework is to provide the underlying distributed event routing and filtering implementations for the selected interaction type plug-ins (see figure 3). The framework can be configured to provide different overlay implementations depending on the network type (e.g. MANET, WAN) and the interaction type of the personality. The overlay plug-in configured for a particular pub-sub personality heavily influences 1) the suitability of the personality to the network environment such as WAN, MANET, 2) scalability and 3) fault tolerance properties of the system. The overlay CF provides pluggable overlays for diverse environments (e.g. probabilistic multicast overlay for MANET and Scribe overlay for WAN etc). The framework is configurable based on environmental context such as the mobility model of the network, for example MAODV[31] overlay implementation can be configured for ad-hoc networks with low mobility pattern and can then be reconfigured to probabilistic multicast overlay if the ad-hoc network becomes highly mobile.

In terms of design, the overlay CFs per-host overlay plug-ins are implemented in terms of three standard component plug-ins (i.e. control, forwarding, state, see figure 3). The *control* component cooperates with its peer brokers on other hosts to build and maintain the broker network topology. It is in charge of managing the overlay event broker network. It encapsulates the distributed algorithms used to establish and maintain the broker overlay structure. The *forwarding* component routes events over

the broker network. This component enables pluggable event forwarding strategies. For example, the simplest approach is to forward the event to all other brokers along the broker tree overlay. Subscriptions are never propagated beyond the broker receiving them. An alternative strategy is subscription forwarding: when a broker receives a subscription from one of its neighbors, it stores the subscription in a subscription table and forwards the subscription to all its remaining neighboring brokers. This effectively sets event forwarding routes through the reverse path followed by subscriptions. Finally, the *state* component encapsulates key states such as nearest broker neighbours lists, connected clients lists (i.e. publishers, subscribers).

In order to test and evaluate the event broker overlay CF, we have populated the framework with three alternative overlay plug-in implementations: probabilistic multicast overlay for configuring pub-sub personality over mobile ad-hoc network, Scribe overlay implementations for wide area networks and IP multicast for local area networks (LAN) and infrastructure based wireless LAN. Furthermore, it is feasible for new overlays to be dynamically integrated into the framework at a later date, in addition to currently available overlay plug-ins.

4 Implementations of GREEN: A Case Study

In this section we consider a pervasive computing application case study and describe how GREEN implementations are configured and reconfigured to meet the requirements imposed by the application and the underlying heterogeneous environment.

4.1 Application Scenario

The scenario embraces vehicular ad-hoc networks (VANET) and wide area fixed network, to facilitate: 1) the autonomous inter-vehicle cooperation over VANET and 2) the monitoring and control of vehicular traffic over a wide area network. The experimental test-bed consists of a small number of robot vehicles augmented with wireless LAN (IEEE 802.11b), GPS, ultrasonic sensors, magnetic compass and on-board PDA. The laptops with WLAN placed on the roadside act as the bridge between the VANET and the Internet. The autonomous vehicles travel along a given path, defined by a set of GPS waypoints (a ‘virtual’ circuit). Every vehicle discovers and cooperates with other vehicles in its proximity to travel safely and avoid collisions. The vehicles in close proximity form a VANET. Furthermore, sensor data generated by vehicles (i.e location, speed, bearings, time stamp) are relayed via WLAN to road-side base stations placed only at strategic points on the road network. These base stations connected to the Internet form a large scale wide area sensor network, thus facilitating traffic monitoring and control. Users in the Internet may query traffic information derived from vehicular sensor data (e.g. slow speed may imply high road traffic). The vehicles approaching a base station needs to temporally reconfigure the personality to operate over from the default WLAN ad-hoc mode to infrastructure mode to relay sensor data to the base station. This application clearly embraces heterogeneous networks and heterogeneous devices and presents two main requirements on GREEN 1) QoS aware event-based middleware suited for mobile ad-hoc networks to enable inter-vehicle communication 2) event-based middleware suited for fixed wide area network to enable dissemination of vehicular sensor data to enable traffic monitoring and control. More details of the VANET test-bed can be found in [28], [29].

4.2 The GREEN Configuration for Mobile Ad-Hoc Networks

The GREEN configuration for MANET is specifically configured to address the following requirements and constraints of the VANET environment:

- support inter-vehicle events communication in a mobile ad-hoc network
- end-to-end event channel QoS monitoring in ad-hoc networks
- content , proximity based interaction and composite events specification

Publish-Subscribe Interaction CF plug-ins for MANET

The GREEN configuration for MANET is illustrated in figure 6. It shows the composition of the component plug-ins within the interaction CF. Similarly it shows the composition of the component plug-ins in the overlay CF. It can be seen the interaction CF is layered on top of the overlay CF by the GREEN top level CF, which is not shown in the figure to simplify the presentation. The publish component (i.e. the one that exports the IPublish interface) and the subscribe component (featuring the ISubscribe interface) publish XML based events and subscribe to events of interest respectively. ISubscribe interface supports a *context* (i.e *proximity*) based subscriptions in addition to topic and content based subscriptions in FEL. The *proximity plug-in* encapsulates the FEL context filter engine extended for location context.

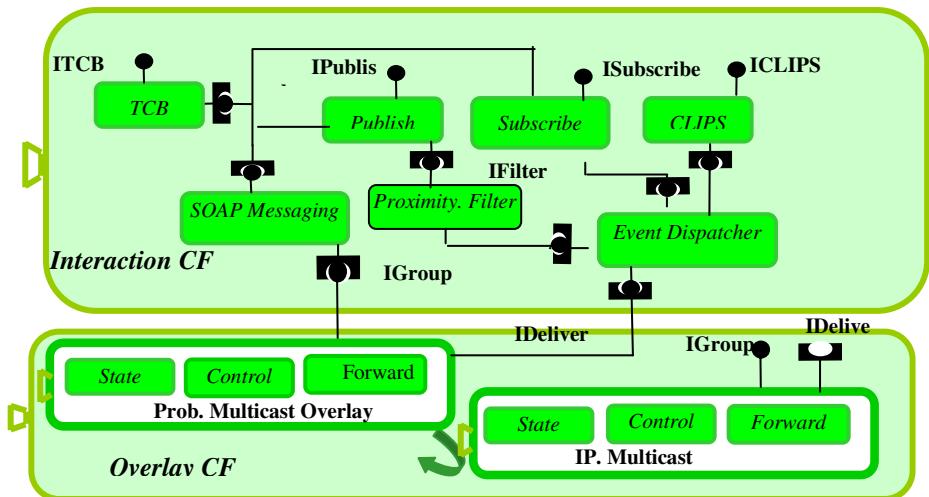


Fig. 6. GREEN configuration for MANET

Furthermore, subscribers can specify *rule-based composite events* using ICLIPS interface. Subscribers specify composite events in the CLIPS language [26] as scripts in a text file and load using the ICLIPS interface. The script files can be (un-)loaded dynamically. The CLIPS component plug-in encapsulates the implementation of CLIPS inference engine. Furthermore, clients can specify *quality-of-service* (QoS) monitoring requirements using ITCB interface, particularly event delivery deadlines and assign them to particular event types (i.e. event channels). For example, ‘event

channel a: event delivery deadline 300ms' and 'event channel b: event delivery deadline 1000ms'. The callback function on the ITCB interface notifies the publisher or subscriber(s) regarding event delivery deadline failures in a guaranteed time bound. The event channel QoS monitoring functionality is implemented by University of Lisboa's *Timely Computing Base* (TCB) [30] plug-in. The TCB plug-in is configured in this personality, as end-to-end event channel QoS monitoring and fail safety is a crucial requirement in VANET for safe driving. Note though that TCB requires a predictable MAC protocol for wireless ad-hoc networks such as TBMAC [32] and also a real-time operating system. Interested readers can refer to [30] for more details on the design of TCB.

Event Broker Overlay CF Plug-ins for MANET

The event broker overlay plug-in must address the unique challenge of MANET environment (i.e. the topology of the network is highly dynamic). There are no fixed infrastructures to place event brokers in MANET. Hence, a fully distributed event broker overlay is implemented; where all mobile nodes perform partial event brokering functionality (i.e event routing, filtering). Notably, producer side and consumer side event filtering is supported. Event producers define the event type and scope of event propagation. Subscriptions (content filters) are deployed only at the consumer side unlike the common approach of *subscription forwarding* [5]. In subscription forwarding strategy, the subscriptions (content filter) form a reverse path for content based event forwarding. However forwarding subscriptions is not suitable in VANET as reverse path(s) quickly become redundant as the network topology is highly dynamic. Hence, in this overlay plug-in, event forwarding is based on event type (i.e. topic) and proximity; each event type is hashed to a underlying multicast group address. Events are forwarded from producers to consumers using the underlying multicast overlay. Scalability has been identified as a drawback with the aforementioned approach in WANs [5], [8]. We address this by allowing producers to define proximity of the event propagation (e.g the proximity radius can be set to 25m), coupled with location aware event forwarding provided top of the underlying multicast overlay. Furthermore, each consumer has to deal with small number of content filters (i.e their own) compared to producers or dedicated event brokers having to match potentially arbitrarily large number of content filters. This helps distribute the event processing overhead evenly among the resource scarce wireless mobile devices and avoids having single points of failure. The aforementioned mechanism adopted by the event broker overlay plug-in is strongly influenced by STEAM [9].

As mentioned before, a multicast overlay plug-in underpins the aforementioned event broker overlay plug-in. This leads to the requirement of designing a multi-hop multicast overlay suited for VANET. There exist numerous multicast algorithms (both proactive and reactive) for ad-hoc networks. However, most existing algorithms (MAODV, AMRoute, CAMP, MCEDAR, etc.) perform inadequately when high node mobility is present in the MANET environment [31] e.g. as in VANETs. Hence we implemented a *probabilistic multicast overlay plug-in* (see fig 6); a multi-hop multicast protocol suited for MANETs with high node mobility. This is an unstructured overlay that intelligently floods events. Each node intelligently decides whether or not each message received should be forwarded to its neighbors. The decision is based on previous messages that the node has received; if a large number of duplicates of a message have already been received, the probability the message is forwarded reduces.

Furthermore, by default, the personality in each PDA (i.e. vehicle) is configured to operate over the probabilistic multicast overlay plug-in (in WLAN ad-hoc mode). The overlay CF is dynamically reconfigured to operate over an IP multicast plug-in (WLAN in infrastructure mode), as shown in fig 6, when the environmental context changes (i.e. ‘if PDA is within the coverage of a fixed base station’).

4.3 A GREEN Configuration for Wide Area Networks

The GREEN configuration for WAN (see figure 7) is configured to address the following requirements and constraints of the fixed WAN environment:

- support events communication in wide area fixed infrastructure based networks to enable distributed vehicular traffic monitoring and control
- support content based interaction type

Publish-Subscribe Interaction plug-ins for WAN

Similar to MANET configuration, the configuration for WAN (see fig 7) shows how the component plug-ins are composed within the interaction CF and the overlay CF to meet the requirements and constraints of WANs. Here we focus only on the differences compared to the MANET configuration. The configuration illustrated in figure 7 allows a *content* based subscription (in addition to topic) only. Assuming content filters would be adequate here, the CLIPS plug-in to specify rule based *composite events* is not configured. WAN is used for disseminating non critical traffic data; hence do not require stringent QoS requirements as in VANET. Therefore, the TCB plug-in is not configured as well.

Event Broker Overlay plug-ins for WAN

The WAN environment requires large scale publish-subscribe middleware between elements across the Internet. In the default configuration shown in fig 7; the event

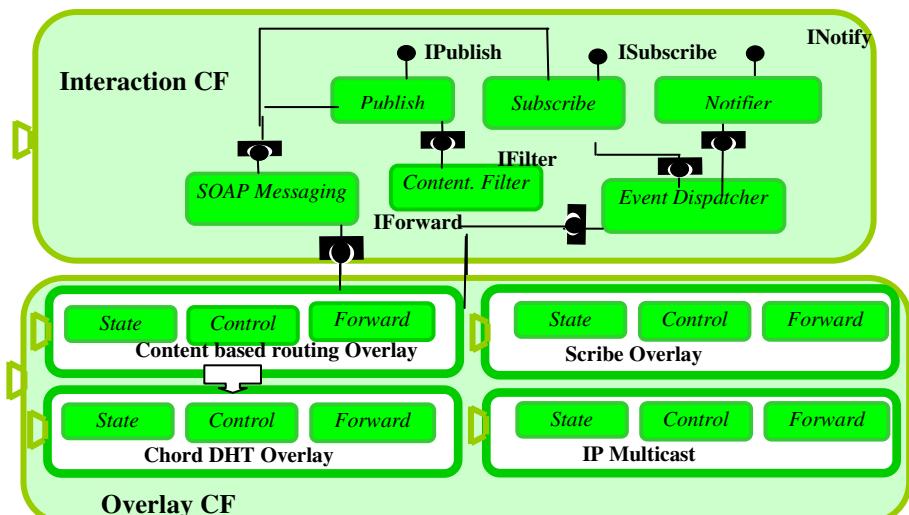


Fig. 7. GREEN configuration for WAN

broker overlay plug-in is underpinned by a Chord DHT (distributed hashtable) [21] overlay.

Here, the event broker overlay plug-in uses rendezvous nodes in the network, which are special event brokers that are known to both producers and consumers. For each event type, a rendezvous node exists in the network. An event type is hashed to a rendezvous point. When a consumer subscribes, the subscription (i.e. content filter) is forwarded towards the rendezvous node R. Every broker that forwards a subscription stores the content filter and event type. Event publications are routed to rendezvous nodes for the event type and then follow the reverse path taken by the subscriptions. This event broker overlay plug-in is similar to the basic event routing mechanisms adopted in Hermes [7]. This overlay plug-in is suited for fixed WANs where the broker topology is fixed. The advantage is the support for content based routing, which is more scalable in WANs. The alternative overlay plug-in as illustrated in fig 7 is Scribe over Chord overlay where each event type is hashed to a Scribe multicast group. This alternative overlay plug-in does not support content based event routing but supports the content based filtering at consumer side. The configuration handles situations where the broker network is subject to topology changes triggered by node and/or link failures. This is made possible as the Scribe overlay [19] manages broker topology changes [19]. Moreover IP multicast plug-in can be configured instead of the Scribe overlay in conditions where network supports IP multicast. A node which is configured to have separate IP multicast and Scribe overlay plug-ins can act as a bridge between the WAN and the VANET (e.g road side base stations mentioned in the application scenario), hence, enabling dissemination of vehicular sensor data generated from VANETs to be distributed over WANs.

Both the above configurations (i.e for MANET, WAN) have been implemented. In addition other configurations, not discussed in this paper, have been integrated to provide publish-subscribe communication infrastructure in our other integrated middleware platforms e.g. CORTEX Middleware for sentient object based, context aware applications in mobile ad-hoc networks [33], ReMMoC a service oriented middleware for mobile clients in infrastructure based wireless networks [34] and GridKit for GRID applications in large-scale networks [18]. The case study clearly demonstrates how the single flexible GREEN middleware is adaptable to various events matching schemes and underlying infrastructure.

5 Evaluation

This section provides concrete performance results of the GREEN family of configurations. It provides quantitative evaluation results on 1) the cost of personality configuration and dynamic reconfiguration and 2) memory footprint cost of GREEN middleware. The experiments utilize a combination of following base device types 1) PDA: HP iPAQ h5450 pocket PC device with a 206MHz strongARM processor, 64Mbytes of system RAM, windows CE 3.0 operating system and IEEE 802.11b wireless network at 11Mbytes/s; 2) PC: 1.7GHz processor and 256Mbytes RAM with windows XP and 100Mbytes/s fast Ethernet.

Experiment 1: Measurements of start-up, configuration, and dynamic fine-grain reconfiguration operations. This experiment evaluates the performance costs incurred by three reflective operations provided by OpenCOM run-time (i.e. loading,

binding, dynamic reconfiguration) on a PDA hosting the GREEN configuration for MANETs (see fig 6 for the configuration). The experiments measure the timing cost of loading components and configuring into configuration A (i.e. IP multicast as the overlay plug-in, see fig 6) and configuration B (i.e probabilistic multicast as overlay plug-in) and finally, the cost of dynamic fine-grain reconfiguration from configuration A to configuration B and vice versa by changing the overlay plug-ins. The results of the experiments are illustrated in fig 8 and demonstrate where the actual overheads occur.

Personality	Components Load time (ms)	Components binding time(ms)	Total start-up time(ms)	Dynamic reconfiguration time(ms)	Total components	Number bindings
Config' A	2580	176	2756	77 (A to B)	10	10
Config' B	2587	171	2758	76 (B to A)	10	10

Fig. 8. Measurements of base reflective operations

The components load time is the most expensive reflective operation and consumes a large part of the overhead incurred in personality configuration. This is because each component is a separate dynamic link library (DLL) that must be first loaded into program memory from storage memory in Windows CE operating system. The component configuration time (i.e. binding time) represents the time taken to initiate a new configuration personality by binding component interfaces to component receptacles. Configuring the middleware personality costs less compared to loading the components (i.e. configuring takes 6.33% of the time compared to 93.66% time taken to load components in config' A personality). Furthermore, the time taken to do fine-grain reconfiguration is consistently lower compared to the initial startup time of the personalities (i.e. fine-grain reconfiguration took 2% of the initial startup time).

Impact of dynamic fine grain reconfiguration. Fig 9 illustrates the experiment investigating the impact of dynamic fine-grain reconfiguration, on a topic-based publish-subscribe service invocation. For this purpose, the middleware was used to invoke 1000 publish calls using both configuration A (i.e IP multicast as overlay plug-in) and configuration B (probabilistic multicast as overlay plug-in) within a host, and dynamically reconfiguring between the two with varying levels of frequency. The first test involved no dynamic reconfiguration; this is a simulated base test of the time taken to perform 500 publish invocations using configuration A and 500 publish invocations using configuration B. Subsequent tests used the architecture meta-model interface of the CFs to dynamically reconfigure the underlying overlay plug-in. In test

Test Description	Time(ms)	Publish calls/second	% Time increase from base test 1
1) 500 publish calls using config A + 500 publish calls using config B	3185	313.97	0
2) 500 config A then 500 B	3283	304.59	3.07
3) 250 config A then 250 B (x2)	3853	259.53	20.97
4) 100 config A then 100 B (x5)	5198	192.38	63.20
5) 50 config A then 50 B (x10)	6260	159.74	96.54

Fig. 9. Cost of dynamic reconfiguration

two, 500 publish calls were performed by configuration A, then dynamically reconfigured to configuration B and then 500 further publish calls were made. Similarly, test three performed 250 publish calls using configuration A then 250 publish invocations using configuration B and this was repeated again.

The results of the five tests are shown in fig 9. It can be seen, as the frequency of reconfigurations increases, the time taken to perform 1000 invocations increases.

For behavior where reconfiguration is generally *out-of-band*, i.e. infrequent compared to the number of base service calls, the additional overhead is less significant (a 3.07% increase in time). However, as fine grain reconfiguration becomes more frequent, e.g. 10 reconfigurations in 1000 base invocations, the overhead becomes significantly greater (a 96.54% increase in time). An example out-of-band scenario which requires the above reconfiguration is, where a PDA is required to reconfigure from configuration A to configuration B when the PDA loses the coverage of a base station. Then the PDA have to use an ad-hoc multicast protocol such as probabilistic multicast overlay instead of IP multicast (i.e. as no support in MANET) to communicate with its peers. Overall the experiment shows dynamic reconfiguration does not necessarily result in high performance cost.

Experiment 2: Evaluation of the memory footprint cost of GREEN

At present mobile and embedded devices have a limited amount of system memory, which can quickly be consumed by the applications. Therefore it is important to minimize the amount of memory needed to store the middleware implementations in a device.

Config-No	Descriptions	Environment
1	Topic based P/S over IP Multicast	WinCE, WLAN
2	Topic based P/S over Prob. Multicast overlay	WinCE, WLAN
3	Content based P/S over IP Multicast	WinCE, WLAN
4	Content based P/S over Prob. Multicast overlay	WinCE, WLAN
5	Proximity based P/S over IP Multicast	WinCE, WLAN , GPS
6	Proximity based P/S over Prob. Multicast overlay	WinCE, WLAN, GPS
7	Config' 3 + QoS (TCB)	WinCE, WLAN
8	Config' 6 + Composite events spec(CLIPS)	WinCE, WLAN , GPS

Fig. 10. Test configurations for memory footprint measurements on PDA

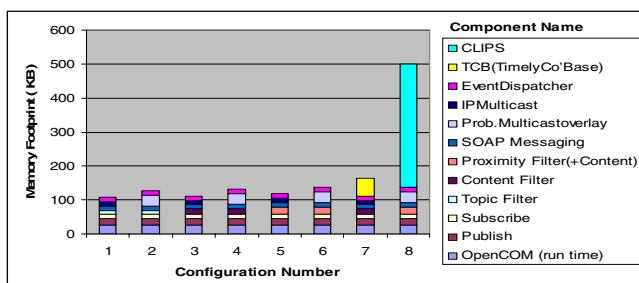


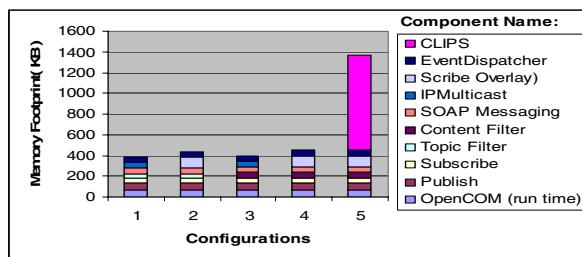
Fig. 11. Memory footprint of GREEN configurations for PDA (MANET)

Config- No	Descriptions	Environment
1	Topic based P/S over IP Multicast	WinXP, LAN
2	Topic based P/S over Scribe overlay	WinXP, WAN
3	Content based P/S over IP Multicast	WinXP, LAN
4	Content based P/S over Scribe overlay	WinXP, WAN
5	Config4+ Composite events spec(CLIPS)	WinXP, WAN

Fig. 12. Test configurations for memory footprint measurements on PC (WAN)

This section examines the resource costs in terms of the static memory footprint of diverse GREEN configurations. Fig 10 documents some valid configurations currently supported for MANETs and fig 11 illustrates the memory costs of the corresponding configurations. Similarly fig 12 documents some valid configurations currently supported for WANs and fig 13 illustrate the memory costs of the corresponding configurations. All the implemented and listed components (in fig 11,13) are OpenCOM components and are implemented in C/C++, except the Scribe overlay component which is implemented in Java. Fig 11 and 13 also illustrate the constituent components of the respective configurations and show how different configurations are composed. The configurations are suited for mobile devices with limited memory, as most configurations for PDAs (WinCE) are around 100Kbytes (e.g. configurations 1, 3, 5 in fig 11). The most expensive configuration in terms of memory for PDAs is configuration 8 and this is mainly due to the high footprint of the CLIPS component. Furthermore, GREEN conserves memory in two distinct levels. Firstly, by only storing the components required by the personality in the storage memory of the device (i.e savings on the storage memory of the PDA). Secondly, only the components that are currently used by configuration are loaded (the OpenCOM run-time provides operations to load and unload components at run-time) into program memory from storage memory (i.e. savings in program memory usage) and components are unloaded from program memory if they are no longer required by the new configuration.

Overall, this experiment shows how GREEN family of configurations achieves low memory footprint despite its generality and flexibility.

**Fig. 13.** Memory footprint of GREEN configurations for PC (WAN)

6 Related Work

Today there exists many research and commercial publish-subscribe systems such as SIENA [5], JEDI [8], Gryphon [6], Elvin [36], MSMQ [37], SonicMQ [38]. This

research has primarily focused on the support of various non-functional properties such as scalability, reliability, etc, largely on fixed network environments. Furthermore, some have attempted to address the emergence of mobile computing: JEDI [8] extends support for client mobility within an infrastructure based wireless networks where event brokers are fixed; and STEAM [9] is specifically designed for mobile ad-hoc networks where broker topology constantly changes. From the functional point of view, existing systems implement a fixed publish-subscribe interaction type (i.e. topic based or content based etc). In general, less emphasis has been placed upon publish-subscribe systems which are open, configurable and re-configurable to support changeable publish-subscribe interactions types which embraces diverse network types and device types.

The only work we know, of constructing highly configurable publish-subscribe middleware are DREAM [39], REDS [40] and the work of Filho et.al [41]. DREAM [39] provides a component framework for configurable and dynamic message-oriented middleware. However, DREAM does not explicitly support distributed network of event brokers. A configurable and dynamic notification service is provided by [41]. However, it does not explicitly support MANETs. REDS [40] provides a configurable, distributed event dispatching system. However it is configurable only within the scope of content based systems in WANs and lacks support for dynamic re-configuration of middleware. Furthermore, these systems do not explicitly support pluggable interaction types and they do not explicitly embrace different network types and device types and hence fall short of providing the level of re-configurability of GREEN.

Finally, there are number of middleware platforms that take a reflective approach to provide configurable and reconfigurable system. However, their focus has largely been on synchronous interaction, e.g. DynamicTAO [42] and UIC [43] are CORBA ORBs offering remote object invocations. Furthermore, there is considerable research in the narrower field of overlay networks themselves but this work is largely orthogonal to our focus. In particular, there are numerous multicast and routing protocols for ad-hoc networks such as MAODV, AMRoute, CAMP, MCEDAR which can underpin different overlays in MANET [31].

7 Conclusions

In this paper we have described our approach to the provision of open, highly configurable and re-configurable publish-subscribe middleware that embraces different network types and interaction types. We have empirically demonstrated using an evaluation scenario: that our architecture, has considerable generality and flexibility in supporting pervasive computing applications. The pluggable event broker overlay structure enables us to embrace different network types such as mobile ad-hoc networks and large scale networks. The pluggable interaction types provide a powerful programming model for the application developers. The architecture is extensible in that new publish-subscribe interaction types and event broker overlays can be developed and plugged into the middleware, even at run-time. Furthermore, performance evaluations of the middleware have demonstrated that flexibility is not necessarily at the expense of performance. Furthermore, the performance figures provide a clear insight into the relative performance tradeoffs for different

configurations. Ongoing work is investigating the impact on GREEN in sensor networks based on motes, tacking the issues of memory size and reconfiguration in such areas.

Acknowledgments

This work is partly supported by the IST-FET-2000-26031, (CORTEX- CO-operating Real-time senTient objects: architecture and EXperimental evaluation) project and FP6-IST-004536 (RUNES-Reconfigurable Ubiquitous Networked Embedded Systems) project.

References

- [1] M. Weiser. Ubiquitous computing. *IEEE Hot Topics*, 26(10):71--72, 1993.
- [2] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri. Generic support for distributed applications, *IEEE Computer*, 33(3):68--76, 2000.
- [3] Blair, G.S., Campbell, A.J., Schmidt, D.C., "Middleware Technologies for Future Communication Networks", *IEEE Network*, Vol. 18, No. 1, January 2004.
- [4] C. Mascolo, L. Capra, Emmerich,w. "Middleware for Mobile Computing (A Survey)". In Advanced Lectures on Networking - Networking 2002 Tutorials, Pisa, Italy. volume 2497 of LNCS, pages 20-58.
- [5] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf "Achieving Expressiveness and Scalability in an Internet-Scale Event Notification Service". Nineteenth ACM Symposium on Principles of Distributed Computing (PODC2000), Portland OR. July, 2000
- [6] G.Banavar et al. An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems. In Proc. of the 19th Int. Conf. on Distributed Computing Systems, 1999.
- [7] P. R. Pietzuch and J. M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In Proc. of the 1st Int. Workshop on Distributed Event-Based Systems, July 2002.
- [8] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Trans. on Software Engineering*, 27(9):827–850, September 2001.
- [9] Rene Meier, V. C. "Steam: Event-based Middleware for Wireless Ad Hoc Networks.". In Proceeding of the International Workshop on Distributed Event-Based Systems (DEBS'02), Austria. 2002.
- [10] Coulson, G., Blair, G.S., Clark, M., Parlantzas, N., "The Design of a Highly Configurable and Reconfigurable Middleware Platform", *ACM Distributed Computing Journal*, Vol 15, No 2, pp 109-126, April 2002.
- [11] Blair, G., Coulson, G., Grace, P., "Research Directions in Reflective Middleware: the Lancaster Experience", Proceedings of the 3rd Workshop on Reflective and Adaptive Middleware (RM2004) co-located with Middleware 2004, Toronto, Ontario, Canada, October 2004
- [12] Clark, M., Blair, G.S., Coulson, G., Parlantzas, N., "An Efficient Component Model for the Construction of Adaptive Middleware", Proc. IFIP Middleware 2001, Heidelberg, Germany, Nov. 2001.
- [13] Coulson, G., Blair, G.S., Grace, P., Joolia, A., Lee, K., Ueyama, J., "OpenCOM v2: A Component Model for Building Systems Software", Proceedings of IASTED Software Engineering and Applications (SEA'04), Cambridge, MA, ESA, Nov 2004.

- [14] Kon, F., Costa, F., Blair, G.S., Campbell, R., "The Case for Reflective Middleware: Building Middleware that is Flexible, Reconfigurable, and yet simple to Use", CACM, Vol. 45, No. 6, pp 33-38, 2002.
- [15] Szyperski, C., Component Software: Beyond Object-Oriented Programming. Addison Wesley, 1998.
- [16] Coulson, G., Grace, P., Blair, G.S., Cai, W., Cooper, C., Duce, D., Mathy, L., Yeung, W.K., Porter, B., Sagar, M., Li, J., "A Component-based Middleware Framework for Configurable and Reconfigurable Grid Computing" to appear in Concurrency and Computation: Practice and Experience, 2005.
- [17] Doval, D ,O'Mahony, D, "Overlay Networks: A scalable alternative for P2P", IEEE Internet computing, jul-aug 2003
- [18] Grace, P., Coulson, G., Blair, G., Mathy, L., Duce, D., Cooper, C., Yeung, W., Cai, W., "GRIDKIT: Pluggable Overlay Networks for Grid Computing", Proceedings of International Symposium on Distributed Objects and Applications (DOA), Larnaca, Cyprus, October 2004
- [19] Castro, M., Druschel, P., Kermarrec, A-M., Rowstron, A., "SCRIBE: A Large-Scale and Decentralised Application-Level Multicast Infrastructure", IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications), 2002.
- [20] Rowstron, A., Druschel, P., "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems", Proc. IFIP Middleware 2001, Heidelberg, Germany, Nov, 2001.
- [21] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications", Proc. ACM SIG-COMM, San Diego, 2001.
- [22] X. Chen, Y. Chen, and F. Rao, "An Efficient Spatial Publish Subscribe System for Intelligent Location-Based Services," Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS '03), June 2003
- [23] S. Schwiderski. Monitoring the behaviour of distributed systems. PhD thesis, University of Cambridge, April 1996.
- [24] A. P. Buchmann. Architecture of active database systems. In N. W. Paton, editor, Active Rules in Database Systems, 2: 29–48. Springer-Verlag, 1999.
- [25] S. Chakravarthy and D. Mishra. Snoop: An Expressive Event Specification Language for Active Databases. Data and Knowledge Engineering, 14(1):1–26, November 1994.
- [26] Gary Riley. CLIPS homepage. <http://www.ghg.net/clips/CLIPS.html>, 2002.
- [27] Charles Lanny Forgy. RETE: A Fast Algorithm for the Many Patterns/Many Objects Pattern Match Problem. Artificial Intelligence, 19(1):17–37, September 1982.
- [28] Sivaharan, T., Blair, G.S., Friday, A., Wu, M., Duran-Limon, H., Okanda, P., Sørensen, C.F., "Cooperating Sentient Vehicles for Next Generation Automobiles", Proc of the MobiSys, 1st ACM Workshop on Applications of Mobile Embedded Systems (WAMES 2004), Boston, USA, June 6, 2004
- [29] Collaborative Robotics Research at Lancaster university <http://www.comp.lancs.ac.uk/computing/users/angie/rendezvous/robotics.html>
- [30] Antonio Casimiro, Paulo Verissimo. Using the Timely Computing Base for Dependable QoS Adaptation. In Pro of the 20th IEEE Symposium on Reliable Distributed Systems, pages 208–217. IEEE Computer Society Press, 2001.
- [31] Royer, E. M., Toh, C-K., A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks, IEEE Personal Communications Magazine, pp 46-55, April 1999.
- [32] R.Cunningham and V. Cahill, "Time Bounded Medium Access Control for Ad Hoc Networks", in Proceedings of the Second ACM International Workshop on Principles of Mobile Computing (POMC'02). Toulouse, France: ACM Press, 2002, pp. 1-8.

- [33] Sørensen, C.F., Wu, M., Sivaharan, T., Blair, G. S., Okanda, P., Friday, A., Duran-Limon, H., "A Context-Aware Middleware for Applications in Mobile Ad Hoc Environments", Proc' of the 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC'2004) at Middleware 2004, Toronto, Canada, October 2004.
- [34] Grace, P., Blair, G. S., Samuel, S., "ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability". In Proceedings of International Symposium on Distributed Objects and Applications (DOA), Catania, Sicily, Italy, November 2003.
- [35] S.Chen, p. Greenfield: QoS evaluation of JMS: an empirical approach, In Proc. of the 37th Hawaii International Conference on System Sciences, Hawaii, USA, 2004
- [36] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelpsosthers. Content Based Routing with Elvin4. In Proc. of the 2000 Australian UNIX and Open Systems Users Group Annnual Conf., Canberra, Australia, June 2000.
- [37] Microsoft Message Queuing (MSMQ),2002. Microsoft, <http://www.microsoft.com/msmq/>
- [38] SonicMQ, 2002. Sonic software, <http://www.sonicsoftware.com>
- [39] M. Leclercq, V. Quema, and J.-B. Stefani. Dream: a component framework for the construction of resource-aware, reconfigurable mom's. In Proc. of the 3rd Workshop on Adaptive and Reflective Middleware, pages 250–255. ACM Press, 2004.
- [40] Gianpaolo Cugola, Gian Pietro Picco. REDS: A Reconfigurable Dispatching System" technical report , Politecnico di Milano (submitted for publications) ,2005
- [41] Silva Filho R. S., De Souza C. R. B., Redmiles D. F. The Design of a Configurable, programmable and Dynamic Notification Service. in Proc. Second International Workshop on Distributed Event-Based Systems (DEBS'03), USA, June 8th, 2003.
- [42] Kon, F., Roman, M., Liu, P., Mao, J., Yamane, T., Magalhaes, L., Campbell, R., "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB", Proc. of Middleware 2000, ACM/IFIP, April 2000.
- [43] Roman, M., Kon, F., Campbell, R., "Reflective Middleware: From Your Desk to Your Hand", IEEE Distributed Systems Online, 2(5), August 2001.)
- [44] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, The many faces of publish/subscribe, ACM Computing Surveys, (2):114--131, 2003.
- [45] XML Path Language (<http://www.w3.org/TR/xpath20/>)
- [46] Grace, P., Coulson, G., Blair, G.S., Porter, B., "Deep Middleware for the Divergent Grid", Proc. IFIP/ACM/USENIX Middleware 2005, Grenoble, France, November 2005.

Transparency and Asynchronous Method Invocation

Pierre Vignéras

GIK Institute of Engineering Sciences and Technology,
Topi, N.W.F.P, 23460 Pakistan
pierre@giki.edu.pk

Abstract. This article focuses on transparency in the context of asynchronous method invocation. It describes two solutions available to provide *full-transparency*: where asynchronism is entirely masked to the developer. The main contribution of this paper is to clearly present the drawbacks of this approach: exception handling and developer consciousness are two problems inherent to *full-transparency* that makes it at least, hard to use, at worst, useless. This paper defends explicit asynchronous method invocation and proposes *semi-transparency*: almost all the complexity of asynchronism is masked to the developer but the asynchronism itself.

Keywords: transparency, asynchronous method invocation, concurrency.

1 Introduction

Transparency is an abstract notion already used in many contexts. Intuitively, its goal is to hide the complexity of an aspect – usually a non functional one – to developers. Java/RMI [16] is a good example of transparency used to hide the complexity of the remote aspect. The syntax of a remote method invocation is almost identical as a local one. The only difference is the fact that remote methods may throw a checked exception (an instance of the class `java.rmi.RemoteException` or of one of its subclass). Anyway, the semantic of such a call is rather different than in the local case: since parameters are marshaled, the remote target of the call gets either copy or references of each original parameters depending on some of their characteristics (primitive type, serializable, implementing the `java.rmi.Remote` interface). This behavior is also used for the result transmission. As seen by the success of the Java/RMI framework, and despite the existence of many works that try to enhance it [13,15,10,17], this example clearly shows that transparency of the remote aspect eases the making of distributed applications. Distributed programming has thus clearly been simplified thanks to transparency. Is it also true in the context of concurrent programming?

Concurrency will probably be one of the major concern for developers in the next decade. Whereas SMP architectures are quite common today, next-generation processors (CMP, SMT, VMT) [1] will provide lots of low-level threads to the operating system. So, concurrency will be almost anywhere, in low-level architectures, in operating systems and in high level languages such as Java and C# which already provide a thread API to express concurrency in object oriented applications.

Asynchronous method invocation is another paradigm that allows the expression of concurrency. It extends very well the standard synchronous method invocation paradigm and also extends naturally to the remote case as remote method invocation does.

This paper focuses on transparency in the context of asynchronous method invocation. Note that the Java language has been used in our study, but many issues described in this paper will also be found in any other object oriented language. Furthermore, some solutions described in this paper is already available in the Mandala project [18].

This paper is organized as follow: section 2 deals with concurrency in the asynchronous method invocation paradigm. Section 3 presents *full-transparency*, and two solutions for its implementation. The reasons why this mechanism should be avoided are also explained in this section. We propose an alternative in section 4. We finally conclude in section 5 along with some perspectives.

2 Dealing with Concurrency

Since we are focusing on transparency, we distinguish the mechanism that provides transparency and the one that provides concurrency. For the latter, many abstractions may be used such as active objects [11], actors [2,3], separates [14], active containers [7,6], or asynchronous references [19].

When making an asynchronous call, the specified method may not be executed concurrently with the caller thread. The underlying abstraction may do many things before running the method while the caller thread may have reached the end of the caller method, or may have already died.

Moreover, when performing many asynchronous method invocations successively, the execution of these methods may also be sequential. This is sometimes necessary when the object on which asynchronous method invocations are made is not designed in a concurrent context (thread-safety, re-entrance). In this case, to prevent problems such as deadlocks and data corruption for example, the underlying abstraction may forbid the concurrent execution of methods using a *non-concurrent asynchronous semantic*: a single thread deal with the method invocation requests. This is the approach of the active object paradigm. On the other extreme, an abstraction may be customized to use a specific *asynchronous policy* (FIFO, one thread per call, thread pool) for the implementation of a given *asynchronous semantic* (non-concurrent or concurrent). This is the way taken by the *asynchronous reference* paradigm.

Nevertheless, the use of any *asynchronous semantic* is subject to deadlocks [19], even non-concurrent one. So even if the asynchronous method invocation may seem simpler to use (compared to thread programming), it does not solve common issues found in concurrent programming in general. Asynchronous method invocation is just a way to *express* concurrency, not a solution to problems it involves. For this purpose, a careful design of classes is still required using concurrent design principles and patterns [12].

3 Full-Transparency

In the case of concurrent programming, we define *full-transparency* as below:

Definition 1 (Full-Transparency).

An asynchronous method invocation is fully-transparent if its syntax is not distinguishable from a standard, synchronous method call. Moreover, the object type used to make an asynchronous method invocation must be compatible with the one used to make a synchronous method invocation.

Two distinct entities must be provided to ensure *full-transparency*:

- the *asynchronous proxy* [8] sends invocation requests to the underlying abstraction¹ which makes the call really asynchronous;
- the *transparent future* [20] used to recover the result.

Transparent futures may use the *wait-by-necessity* mechanism [4] provided by ProActive [5] and Mandala [18]: when a client makes an asynchronous call, a future object – subtype of the original type declared by the method – is immediately returned. When the client uses this future, it is blocked until the real result becomes truly available. The figure 1 illustrates the mechanism: a client calls a method `p.m()` on an asynchronous proxy (1). This last uses an abstraction to realize the actual asynchronous invocation (2) which may lead to the concurrent execution of the method `m()` (4'). The *future* returned by the abstraction (3) is then wrapped into a *transparent future* which is a subtype of the original result. Client can thus use the result, `r`, as usual thanks to polymorphism. When a method, say `foo()` is called on the result `r` (4), the *transparent future* uses the wrapped *future* to know the status of the asynchronous method invocation through the call `waitForResult()`² (5). When the real result is available (5'), the original call `foo()` is finally invoked (6).

Note that the property about *type compatibility* may produce concurrency where it is not expressed explicitly: by the passing of a type-compatible asynchronous proxy to a library, the method invocation made by the latter on the former, while expressed synchronously, may execute concurrently. Hence, allowing the use of legacy classes in a concurrent context may ease the production of concurrent applications: this is the main goal of *full-transparency*.

3.1 Solutions in Java

Two solutions may be provided to implement fully-transparent asynchronous method invocations in Java. They both use polymorphism in method invocation.

Using inheritance: The ProActive approach. The first solution, used in the ProActive framework, is based on inheritance. Basically, an asynchronous proxy is an instance of a class which extends the original object class. A transparent future is an instance of a class which extends the one returned by the original method. For instance, after an object, say `a`, instance of a class `A`, becomes active, client uses a *proxy* `p` which class

¹ The abstraction and the asynchronous proxy may be the same object, but in this paper we distinguished the two, as the asynchronous proxy is the one which actually provides transparency.

² The `java.util.concurrent.Future.get()` method has the same functionality, but we keep `waitForResult()` in this article which is more explicit.

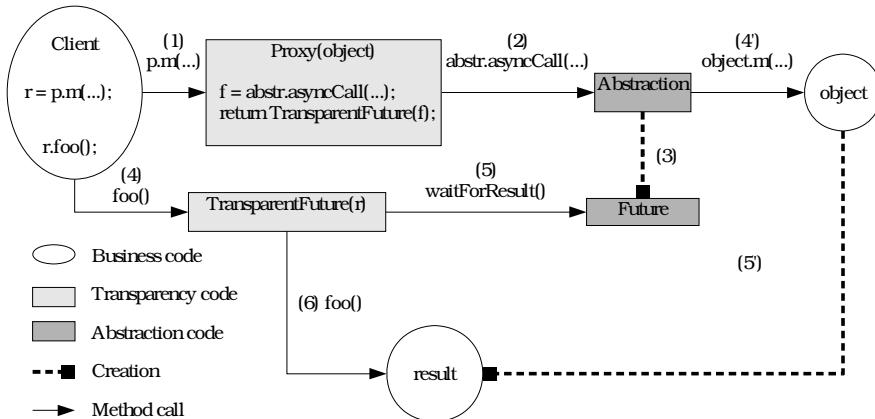


Fig. 1. Illustration of a *fully-transparent* asynchronous method invocation on an *asynchronous proxy*

P extends A. Each method of A is redefined in P. So, if A declares: `T m(...)`, then in the proxy class P, the method `m()` returns a transparent future which type is a subtype of T. This subclass implements the *wait-by-necessity* behavior: each call runs a test to know the availability of the result. If the test is true, the real method is invoked, else the client is blocked. When the result becomes available, every blocked threads are notified.

This solution contains many problems, at least in Java:

- final classes cannot be inherited preventing the use of both asynchronous proxies and transparent futures (on 3,950 classes of the JDK v1.4³, only 57% are declared public on which 9% are in this case);
- final methods (5% of public methods of public classes) cannot be overridden preventing the implementation of both asynchronous proxy and transparent futures;
- methods returning a primitive type (27% of public methods of public classes) cannot be overridden to return a subtype used by the *wait-by-necessity* mechanism;
- clients accessing to public fields of a transparent future cannot be blocked waiting the availability of the result of the asynchronous method invocation.

The last problem may seem minor since, as a common good practice, public fields are usually also declared static final: accessing such fields is not a problem neither in a concurrency context nor in a distributed context. Anyway, if 99.9% of class fields in the JDK v1.4 are declared final, only 20% of instance fields are too: 8 non-final public instance fields are found for any 100 public classes found.

Since those problems are directly related to inheritance, another approach may avoid them.

³ The Java example program, called `ClassPathAnalyser`, and released with the Mandala framework [18] has been used. Only classes with a full class name prefixed by `java` was considered.

Using interfaces: The Mandala approach. Java interfaces do not contain fields (or they are constant) and their methods are all declared (implicitly) public. Moreover, they cannot be declared final. Hence, the exclusive use of interfaces for full-transparency of asynchronous method invocation solves problems found in the inheritance solution.

Java provides *dynamic proxy* since the JDK v1.3 : the `java.lang.reflect.Proxy` is able to produce a class at runtime implementing a given set of interfaces. The business code of the proxy is an instance of a class which implements `java.lang.reflect.InvocationHandler`.

Using the dynamic proxy feature, it is possible to implement a fully-transparent asynchronous method invocation mechanism. First, an *asynchronous proxy* can be defined using the general code design of PROG 3.1.

```

1  public class AsynchronousProxy implements InvocationHandler {
2      // Creates concurrency
3      private Abstraction abstraction;
4
5      /**** InvocationHandler implementation ****/
6      public Object invoke(Object proxy, Method method, Object[] args)
7          throws Throwable {
8          Class returnType = method.getReturnType();
9          Class[] resultInterfaces;
10         if (returnType.isInterface()) {
11             resultInterfaces = new Class[] {returnType};
12         }else{
13             resultInterfaces = returnType.getInterfaces();
14         }
15         Future future = abstraction.asyncCall(method, args);
16         return Proxy.newProxyInstance(resultInterfaces,
17                                         new FutureProxy(future));
18     }
19 }
```

PROG. 3.1. full-transparent implementation of an asynchronous proxy using dynamic proxies

In this code, the concurrency is produced by a supposed abstraction able to invoke asynchronously a given method⁴. This invocation (supposed made by the `asyncCall()` method) must return a `Future` instance such as the one found in the `java.util.concurrent` of the new JDK v1.5. This object is then encapsulated in a `FutureProxy` instance class which is returned. This class is a subtype of the original result, thanks to the use of dynamic proxies another time. The business code of this proxy implements the *wait-by-necessity* mechanism with a code similar to PROG 3.2.

As seen, each method called on our *transparent future* waits for the actual return of the underlying asynchronous method call, and redirects the call to the business object.

⁴ Reflection is used in this case.

```

1 class FutureProxy implements InvocationHandler {
2     final Future future;
3
4     FutureProxy(final Future future) {
5         this.future = future;
6     }
7     /*** InvocationHandler implementation ****/
8     public Object invoke(Object proxy, Method method, Object[] args)
9         throws Throwable {
10        // Object's method must not be redefined.
11        if (method.getDeclaringClass().equals(Object.class)) {
12            return method.invoke(this, args);
13        }
14        Object result = future.waitForResult();
15        return method.invoke(result, args);
16    }
17 }
```

PROG. 3.2. Implementation of the *wait-by-necessity* mechanism in the business code of a future dynamic proxy

The major drawback of the approach based on interfaces is the constraints that limit its use of application:

- the object used must be of a class which implements at least one interface;
- the method invoked asynchronously must be defined in an interface;
- the return type of the method must also be an interface.

Among the 57% declared public classes over the 3,950 found in the JDK v1.4⁵, 20% are interfaces and 30% implement at least one interface. Only 5% of public methods are declared in interfaces. Few of them return a type which is either defined by an interface or a class which implements an interface: over 17,254 public methods of the JDK v1.4, only 653 (4%) conform to the previous criterion and are thus usable with a fully-transparent asynchronous proxy based on interfaces. It is then clear that this solution rarely allows the use of such proxies in applications not designed for it.

3.2 Inherent Problems

One of the goal of full-transparency, is to allow the use of legacy classes which were not designed in a concurrent context. Polymorphism is the core of the mechanism: the use of subtypes (either by inheritance or by interface) allows the passing of *asynchronous proxy*, and *transparent futures* to some methods which believe they are standard objects. This *may* naturally produce concurrency. This section shows that full-transparency has inherent problems which make it at best, difficult to use, at worst useless.

⁵ Only classes with a full class name prefixed by java were considered.

Exception Handling. When considering legacy code, the instructions given in PROG 3.3 are commonly found in a Java program. If the call `out.write(b)` is asynchronous (and

```

1 int b = ...; // byte to write
2 java.io.FileOutputStream out = null;
3 try{
4     out = new java.io.FileOutputStream(...);
5     out.write(b);
6 }catch(java.io.IOException ioe) {
7     // handle any IO exception
8 }catch(java.lang.SecurityException se) {
9     // handle security exception
10 }
```

PROG. 3.3. full-transparency and exceptions

of course fully-transparent), then the caller thread continues its execution and may reach a point far beyond the `catch()` statements when the `write()` method actually ends.

In the case of checked exceptions, such as the `java.io.IOException`, one approach (the ProActive one) is to make these calls synchronous. This trivially prevents the problem to appear, but does not solve the case of unchecked exceptions such as the `java.lang.SecurityException`. A solution, is then to enforce a synchronization between the caller and the callee thread by using the future such as in the PROG 3.4:

```

1 java.util.List list = library.getAList();
2 ...
3 try{
4     // Asynchronous call
5     Object o = list.remove(0);
6     // *Must* wait the result (in case of a runtime exception) !
7     o.toString();
8 }catch(UnsupportedOperationException e) {
9     // This exception is a runtime exception
10 ...
11 }
```

PROG. 3.4. Explicit synchronisation

This may be done by an automatic tool, or at least by a checker. But even in this case, the method `write()` of the first example is declared returning `void`. So there is no way to enforce clients to use a non-existent result!

This problem is still open. Whereas, some directions are work in progress in our team, we believe it is a major drawback that makes full-transparency very hard to use when exceptions are considered.

Developer Consciousness. As for us, the main problem of the full-transparent mechanism is related to the implicit concurrency it seems to produce. The degree of concurrency in an application written using the asynchronous method invocation paradigm can be increased by following the general guidelines:

Definition 2 (Guidelines for efficiency).

A maximum concurrency degree is achieved using an asynchronous method invocation paradigm when:

- *method calls are made as soon as possible;*
- *result recovery are used as late as possible.*

This may seem trivial but it is not the natural way applications are written in the sequential world. Consider the following code:

```

1 // Step 1
2 MyClass myObject = new MyClass();
3 // Step 2
4 MyInfos infos = myObject.myMethod(myParameters).getInfos();
5 // Step 3
6 doSomething();
```

This code scheme is very common in Java. If the `myObject` variable references a fully-transparent asynchronous proxy, the resulting code will not be more efficient. It will even be less efficient since the handling of concurrency has a cost. Hence, to gain in concurrency, the code must be rewritten:

```

1 // Step 1
2 MyClass myObject = new MyClass();
3 // Step 2
4 MyResult result = myObject.myMethod(myParameters);
5 // Step 3
6 doingSomething();
7 // Step 4
8 MyInfos infos = result.getInfos();
```

And this is clearly not a natural sequential code in Java.

Hence, full-transparency does not allow the becoming concurrent of sequential application almost automatically. Most of the code must be rewritten – at least reordered – to gain some efficiency. These modifications may be done by a tool (more or less automatic), but in this case, why focusing on full-transparency?

We strongly believe asynchronous method invocation should be as simple as its synchronous version. But it must be *explicit* in order to ensure the writing of efficient concurrent applications.

4 Proposition: Semi-transparency

Following the conclusion of the previous section, we call *semi-transparency*, the mechanism which masks almost every aspect of the concurrency involved by an asynchronous method invocation (abstraction, asynchronous semantic and policy) but the asynchronous itself. This mechanism defends explicit expression of concurrency as with the `java.lang.Thread` API in opposition to the implicit concurrency provided by a fully-transparent solution.

Hence we propose the following syntax – promoted in Mandala:

Notation 1 (Semi-transparency syntax).

If a public method has the following signature:

$T\ m(A_1\ a_1, \dots, A_n\ a_n)\ throws\ E_1, E_2, E_n$

then its semi-transparent asynchronous version has the signature:

$\text{Future} <T>\ \#m(A_1\ a_1, \dots, A_n\ a_n, \text{Meta}\ \text{meta})$

In particular, exceptions declared in $m()$ are no more part of the signature of $\#m()$. The object meta may contain some informations used by the underlying abstraction such as priority, before and after methods, security informations, etc. It must at least contain an exception handler which will be used by the underlying abstraction when an exception occurs.

This syntax solves many problems:

- strong typing is ensured (thanks to generics);
- the developer knows the asynchronous nature of the invocation of $\#m()$ ⁶ thanks to its signature which differs from the original;
- exceptions are always handled by a *client specific* object⁷ and thus can never be ignored; anyway, exceptions may be re-thrown on the client side when retrieving the result through the return future.

The last problem to solve is where these method will be found? Which class defines them?

4.1 Asynchronous Views

Definition 3 (Asynchronous View).

The asynchronous view of a class C – noted $\text{view}(C)$ – defines, for each public method $m()$ in C , its semi-transparent asynchronous version $\#m()$. If C is an interface, then

⁶ Even if semi-transparency is provided in Mandala [18], the '#' character is reserved in Java and cannot be the first of an identifier (field or method). This character is replaced by the prefix `rami` where RAMI stands for Reflective Asynchronous Method Invocation.

⁷ Exceptions are always handled though. But the default handler found in the `ThreadGroup` class is not a good solution. Consider remote asynchronous method invocation as a case study.

for each method $m()$ declared in C , its semi-transparent asynchronous version $\#m()$ is also declared in $view(C)$.

If B is a supertype of C , then $view(B)$ is also a supertype of $view(C)$. Anyway, $view(C)$ is not a subtype of C .

Furthermore, an instance of an asynchronous view is called a semi-transparent asynchronous proxy.

As for the naming of asynchronous view, we propose a mirroring of the standard Java class naming: suppose a full class name is $p.s.C$, then using a prefix, $jaya$ ⁸, the full asynchronous view name (a class) is: $jaya.p.s.C$. This enforce the developer to be conscious of its use of asynchronous views (since they are really distinct classes) and so, to follow the guidelines given in definition 2. Note that using a naming convention which is just based on the class name, such as $p.s.Async_C$ or similar, prevents its use in the standard Java language: some packages may be *sealed* preventing the addition of new classes.

The generation of asynchronous views leads naturally to a hierarchy of types which is symmetric from the original. As an example, consider the asynchronous view generation of the standard class `java.io.FileWriter`. The figure 2 presents the UML class diagram⁹. The left part are the asynchronous views hierarchy which clearly mirrors the type hierarchy of their related class on the right. Hence, the `java.lang.Object` superclass, has its asynchronous view symmetric called `jaya.java.lang.Object`. This view plays an identical particular role: it is a supertype of any other asynchronous view.

An asynchronous view also provides synchronous methods. Consider a method $m()$ defined in a class $p.C$. Then, it also exists in $jaya.p.C$. But the semantic of methods $C.m()$ and $jaya.p.C.m()$ are really different: the latter must be a shortcut for $jaya.p.C.\#m().waitForResult()$. The reason is that each call made on an asynchronous proxy – an instance of a view – must have reached the underlying abstraction used. Consider an abstraction which provides both concurrent and remote aspect as a case study¹⁰. Synchronous version of methods are clearly important to prevent developers from mixing standard, synchronous classes with asynchronous views. As a side effect, the class naming convention prevents even more this mixing. The fact that asynchronous views and standard classes short names are homonyms forbid developers to use both without care. For example, the following instructions are ambiguous and does not compile:

⁸ Since the prefix is arbitrary, the name `jaya` was chosen for its meaning in Sanskrit (“Victory”) where the Mandala name also comes from. Moreover, the asynchronous views generator of Mandala is called `jayac` which sounds like `javac`.

⁹ This diagram has been produced from a real code generation thanks to the `jayac` asynchronous view generator of the Mandala framework. This is the reason why asynchronous methods are prefixed by ‘`rami_`’ instead of the character ‘`#`’.

¹⁰ As provided by the *stored object reference* [6], an extension of the *asynchronous reference* paradigm [19], that uses the *active container* concept [7] to provide the remote aspect.

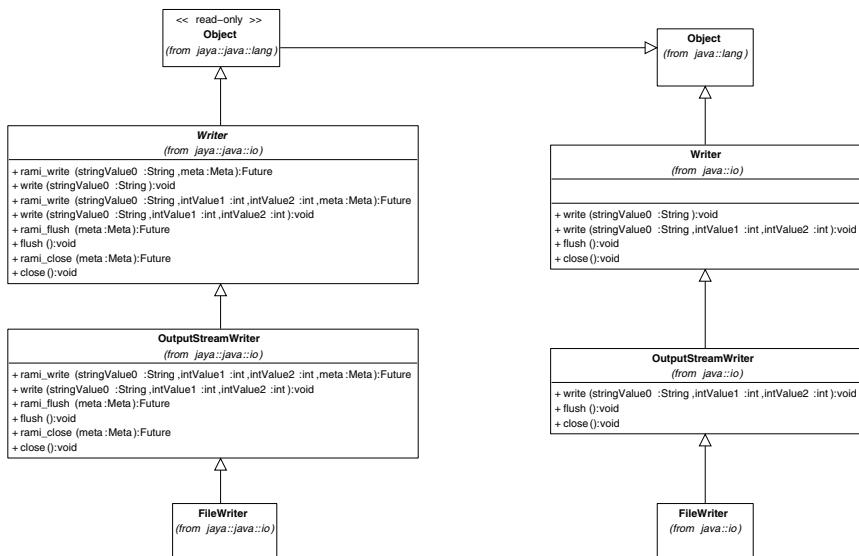


Fig. 2. Type hierarchy of the asynchronous view generation of the `java.io.FileWriter` class

```

1 import java.io.*;
2 import jaya.java.io.*;
3 ...
4     Writer writer = new FileWriter("foo.txt");
5     writer = new FileWriter("bar.txt");
  
```

So the developer is enforced to use a full class name. If most of the code is synchronous, he would write:

```

1 import java.io.*; // Use shortcuts for synchronous class only
2 ...
3     Writer writer = new FileWriter("foo.txt");
4     jaya.java.io.Writer writerProxy =
5         new jaya.java.io.FileWriter("bar.txt");
  
```

On the opposite, when most of the code is asynchronous, he would prefer the following form:

```

1 import jaya.java.io.*; // Use shortcuts for asynchronous view only
2 ...
3     java.io.Writer writer = new java.io.FileWriter("foo.txt");
4     Writer writerProxy = new FileWriter("bar.txt");
  
```

This leads to a naturally cleaner code where the developer knows it is using an asynchronous proxy. Hence, it enforces him to follow the guidelines 2, and allows him to enhance the overall concurrency degree of the whole application.

5 Conclusion

This article focuses on mechanisms used to hide the complexity of asynchronous method invocations. We have seen that this paradigm may use many underlying abstractions to handle concurrency: active objects, actors, separates, active containers, asynchronous references are several options among others. We show that *full-transparency* may be provided using two solutions: inheritance or interfaces. The former contains several problems the latter solves by imposing very high constraints. Finally, while the main advantage of full-transparency is its possible application with legacy code, it has two inherent problems that make its use very complex as far as exceptions is concerned or useless as far as efficiency is concerned. So we defend an explicit expression model which masks the most of the asynchronism mechanism, and abstractions in particular, but the asynchronism itself. This model is called *semi-transparency*. It provides a solution to the exception problem. It also helps the developer to focus on the concurrent aspect of its application. This enforce the following of guidelines given in definition 2 which seems necessary to gain the most of concurrency.

Transparency, both *full* and *semi* is proposed in the Mandala framework [18] which helps the development of concurrent (and eventually distributed) Java applications. As such, the framework must be extended to use the new *java.util.concurrent* package. Furthermore, exceptions handling in the context of both full- and semi-transparent asynchronous method invocation must be further studied.

References

1. TLP and the Return of KISS . Web page, January 2004.
<http://www.aceshardware.com/read.jsp?id=60000312>.
2. AGHA, G. *Actors: A Model Of Concurrent Computation In Distributed Systems*. PhD thesis, University of Michigan, 1986.
3. AGHA, G., AND HEWITT, C. Concurrent programming using actors: Exploiting large-scale parallelism. In *Readings in Distributed Artificial Intelligence*, A. H. Bond and L. Gasser, Eds. Kaufmann, San Mateo, CA, 1988, pp. 398–407.
4. CAROMEL, D. Toward a method of object-oriented concurrent programming. *Communications of the ACM* 36, 9 (1993), 90–102.
5. CAROMEL, D., KLAUSER, W., AND VAYSSIÈRE, J. Towards seamless computing and meta-computing in Java. In *Concurrency: practice and experience* (Sept.-Nov. 1998), G. C. Fox, Ed., vol. 10, Wiley and Sons, Ltd., pp. 1043–1061.
6. CHAUMETTE, S., AND VIGNÉRAS, P. A framework for seamlessly making object oriented applications distributed. In Joubert et al. [9], pp. 305–312.
7. CHAUMETTE, S., AND VIGNÉRAS, P. Behavior model of mobile agent systems. In *FCS'05 - The 2005 International Conference on Foundations of Computer Science* (Las Vegas, USA, june, 27–30 2005). H. R. Hamid and R. Joshua, Eds., CSREA Press. ISBN: 1-932415-71-8.
8. GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. ISBN : 0-201-63361-2.

9. JOUBERT, G. R., NAGEL, W. E., PETERS, F. J., AND WALTER, W. V., Eds. *Parallel Computing: Software Technology, Algorithms, Architectures and Applications, PARCO 2003, Dresden, Germany* (2004), vol. 13 of *Advances in Parallel Computing*, Elsevier.
10. KURZYNIEC, D., WRZOSEK, T., SUNDERAM, V., AND SŁOMIŃSKI, A. RMIX: A multi-protocol RMI framework for java. In *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS'03)* (Nice, France, Apr. 2003), IEEE Computer Society, pp. 140–146.
11. LAVENDER, R. G., AND SCHMIDT, D. C. Active object: an object behavioral pattern for concurrent programming. *Proc. Pattern Languages of Programs*, (1995).
12. LEA, D. *Concurrent Programming in Java. Second Edition: Design Principles and Patterns*. Addison-Wesley Longman Publishing Co., Inc., 1999.
13. LYON, D. CentiJ: An RMI Code Generator. *Journal of Object Technology* 1, 5 (November-December 2002), 117–148.
http://www.jot.fm/issues/issue_2002_11/article2.
14. MEYER, B. Systematic concurrent object-oriented programming. *Communications of the ACM (special issue, Concurrent Object-Oriented Programming, B. Meyer, editor)* 36, 9 (1993), 56–80.
15. NESTER, C., PILIPPSEN, M., AND HAUMACHER, B. A more efficient RMI for Java. In *Proceedings of Java Grande Conference* (San Francisco, California, june 1999), ACM, pp. 152–157.
16. SUN MICROSYSTEMS. *Java Remote Method Invocation Specification*, 1998.
<http://java.sun.com/products/jdk/1.1/docs/guide/rmi/>.
17. THIRUVATHUKAL, G. K., THOMAS, L. S., AND KORCZYNSKI, A. T. Reflective remote method invocation. *Concurrency: Practice and Experience* 10, 11–13 (1998), 911–925.
18. VIGNÉRAS, P. Mandala. Web page, August 2004.
<http://mandala.sf.net/>.
19. VIGNÉRAS, P. *Vers une programmation locale et distribuée unifiée au travers de l'utilisation de conteneurs actifs et de références asynchrones*.
Rapporteurs : Doug Lea, Françoise Baude, Michel Riveill.
Jury : Françoise Baude, Serge Chaumette, Olivier Coulaud, Mohamed Mosbah, Alexis Moussine-Pouchkine, Michel Riveill. PhD thesis, Université de Bordeaux 1, LaBRI, november, 8th 2004.
<http://mandala.sf.net/docs/thesis.pdf>.
20. WALKER, E., FLOYD, R., AND NEVES, P. Asynchronous Remote Operation Execution In Distributed Systems. In *International Conference on Distributed Computing Systems* (Paris, France, May/June 1990), no. 10, pp. 253–259.

CORB: A Controlled Resource Borrowing Framework for Overload Handling in Cluster-Based Service Hosting Center^{*}

Yufeng Wang, Huaimin Wang, Dianxi Shi, and Bixin Liu

National University of Defense Technology, ChangSha 410073, China
{yfwang, dxshi, bxliu}@nudt.edu.cn, whm_w@163.com

Abstract. The paper proposes a resource framework *CORB* for overload handling in component application hosting center through dynamic resource borrowing among hosted applications. The main idea is to utilize the fine-grained idle server resource of other applications to partake of surging workload, while keeping the resource borrowing under control for not violating the SLA of the donor application. The contribution of the paper is two-fold: (1) a queuing analysis-based resource borrowing algorithm is proposed for overloaded applications to acquire as exact amount of resource as possible; (2) an adaptive threshold-driven algorithm is presented to drive the overload handling with threshold values adaptively tuned according to changing workload. Empirical data is presented to demonstrate the efficacy of CORB for overload handling and response time guarantee in a prototype service hosting cluster environment.

1 Introduction

The growing cost of owning and managing computer systems is leading to outsourcing of commercial services to hosting centers. These centers usually provision cluster of servers to multiple applications under certain SLAs. Performance guarantee has been the primary concern in the design and operation of the cluster-based hosting center, especially under Internet workload conditions. The uncertainty of clients' scope and visiting patterns makes burstiness a fundamental property of workload of Internet computing systems, and be observed across all time scales [1]. Dynamically surging workload is becoming the main cause of service degradation and server overload. Unfortunately, most hosting centers have not adequately addressed the management of extreme load, relying mainly on overprovisioning of resources or load shedding. On the other hand, statistical observations show that most servers are underutilized in hosting data center environment. For example, the utilization of nearly 80% of thousands of servers of HP Data Center is below 30% [2], which further manifests the inconsistency between the resource provisioning needs for overload handling and general resource underutilization. To address this

* This research was partially supported by the National Basic Research Program (973) of China (No.2005CB321804), the National Natural Science Foundation of China (No.90412011), the National Hi-Tech Research and Development Program (863) of China (No.2004aa112020).

inconsistency, dynamic server allocation techniques have been proposed to better utilize the resources in flash crowds conditions [3]. These dynamic allocation schemes react to changing application loads by reallocating resources to overloaded applications in the granularity of server node. However, server-granularity resource allocation cannot fully utilize the hosting center resources [4]. This paper describes our initial work on a fine-grained resource borrowing framework *COROB* for component-based application hosting center. The framework can not only help attack partial application overload problem by utilizing unused server resource among applications, but also provide service level agreement guarantee for donor applications who share the under-utilized server nodes. While most of the recent studies have focused on the overload protections and resource provisioning, there is little investigation of feasibility and methods of fine-grained resource borrowing for performance management in the presence of partial overload. Compared with existing hosting center performance management techniques, our contributions are two-fold. First, a queuing analysis-based resource borrowing algorithm is proposed for overloaded applications to acquire as exact amount of resource as possible. Secondly, we design a dynamic threshold-driven algorithm to drive the overload handling for hosting center with threshold values adaptively tuned adapting to workload change.

The paper is organized as follows. Section 2 gives background of component application hosting center and two kinds of hosted application overload. Section 3 describes the *COROB* resource borrowing framework and presents how to choose nodes for an overloaded application with the donor's performance guaranteed. Section 4 presents the threshold-based overload handling algorithm. Then in section 5, some experimental results from a prototype of service hosting environment are presented and discussed. After section 6 provides an overview of related work, section 7 concludes the paper.

2 Background

Most hosting centers are organized into server cluster to achieve higher scalability and availability. Although component application hosting environments are always presented in multi-tier architecture, this paper mainly focuses on resource borrowing and overload handling in business logic layer, which is built on the application server cluster. Throughout the paper, we refer to the component applications hosting infrastructure simply as *cluster center*, which is composed of cluster of application servers and provides the management capability such as deployment, workload distribution, performance monitoring and overload handling. We refer to a hosted component application deployed on parts of the servers as *cluster application*. While the resource of the cluster center could be multiplexed by more than one application, each cluster application may behave to its clients as if the whole cluster center were exclusively occupied by itself. Fig 1 sketches the relations between cluster center and hosted cluster applications and Fig 1-(c) represents the case that two cluster application share cluster node N3. We refer to the application who offers the under-utilized nodes as *donor application*, and the one who borrows the node as *borrower application*. The borrowed node will be shared and process requests from both the donor and the borrower applications. The cluster center supports server node

donation, borrowing and performance guarantee for donor application through a resource borrowing framework called *COROB*, which will be presented in detail in section 3.

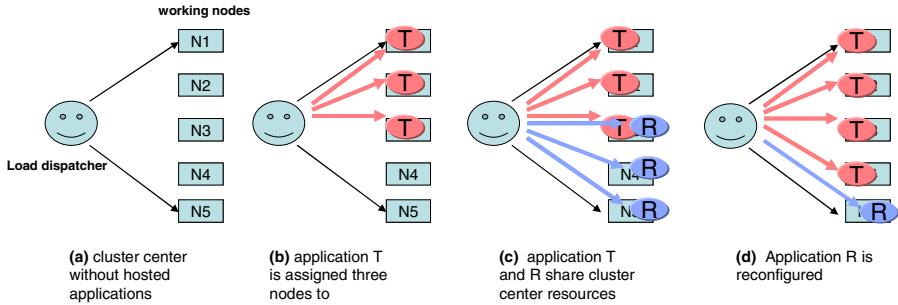


Fig. 1. Cluster center and hosted cluster applications

Consider a cluster center consisting of N server nodes denoted by $P = \{n_i | 1 < i <= N\}$ and M cluster applications denoted by $CA = \{ca_j | 1 < j <= M\}$. Formally, we model a cluster application j as $ca_j = (Components_j, Config_j, SLA_j)$, where $Components_j$ denotes the components set of ca_j , $Config_j$ denotes the configuration structure of ca_j , and SLA_j denotes the service level agreement of ca_j . P_j denotes the cluster server nodes set assigned to ca_j . For simplicity, we make following assumptions throughout the paper. Cluster applications use components fully duplicating strategy, i.e. each server node of an application has the same computing capacity and components configuration initially. CPU resource is the bottleneck, which is always the case for application server, and workloads of ca_j are equally distributed among cluster server nodes in P_j .

Once a cluster application has been deployed, its processing capacity is decided and indicated by its response time and maximum throughput, expectation of which are typically specified in SLA to express its quality of service requirements. Queuing theory shows that with the increasing workload, an application response time will increase sharply at some points because of longer queuing time. And system throughput will ultimately reaches its limit as the bottleneck resources of the system saturate at nearly 100% utilization. After saturation point, system performance begins to degrade because of severe resource contentions [3]. In session-oriented load distribution scheme which is representative in most of Web and component cluster center, a session setup request first arrives at a front-end load dispatcher and then is redirected to one of back-end working servers according load balancing strategies such as round robin. All following requests in this session will go directly to this working node. So we model session-oriented workload of an application as $\lambda = \lambda_{sessions} \times \lambda_{request-per-session}$, in which $\lambda_{sessions}$ denotes the session rates and can be regarded as workload on load dispatcher and $\lambda_{request-per-session}$ denotes the requests arrival rates within a session. Base on this understanding, we can classify cluster application overload into two classes: *local working node overload* and *whole application overload*:

- The overload caused by some suddenly increasing $\lambda_{request-per-session}$ is referred to as local working node overload.
- The overload caused by surging sessions $\lambda_{sessions}$ of an application is referred to as whole application overload.

This classification details two scenarios of cluster application overload and can help adopt differentiated overload handling techniques. In facts, statistical observations show that most servers are underutilized in today hosting data center. This paper aims to provide a dynamic resource borrowing framework to handle partial application overload.

3 COROB: A Controlled Resource Borrowing Framework

A novel fine-grained resource management framework called **C**Ontrolled **R**e*s***O**urce Borrowing framework (or *COROB*) is presented in this section. With COROB, an application can share its underutilized server nodes in a controlled way to help relieve other overloaded applications.

3.1 Controlled Resource Borrowing

This paper assumes SLAs are used to apparently specify whether an application is willing to share its under-utilized nodes, as well as the conditions of resource donation such as when to begin sharing nodes and the constraints to be met during resource sharing. For example, the following SLA segment says that the application supports server nodes sharing only if the time of day is between 06:00 and 20:00 and the CPU utilization is less than 20%, and during resource sharing CPU utilization must be kept below 90% and average response time must be guaranteed less than 150 ms.

SUPPORT SHARING, IF 06:00 <TIME<20:00 **and CPU utilization < 20%, with**
CONSTRAINTS CPU utilization < 90% and response time < 150ms.

As the donor applications may take the risk of performance disturbance because of resource donation, mechanisms should be designed to compensate the donors and encourage resource sharing. To address this issue, incentive mechanisms have been proposed [14]. In this paper, we focus on how to meet the resource donation constraints and provide SLAs guarantee, which is a challenging issue that must be well addressed during the resource borrowing framework design. In COROB, we carry out a queuing theory analysis for cluster applications, based on which a resource borrowing algorithm is proposed for overloaded applications to acquire as exact amount of resource as possible. On the borrowed nodes, we periodically calculate the mean request arrival rate limits for the borrower application, and through controlling the admission rates of the borrower and the donor one, we can statistically guarantee the average response time of the donor.

Consider server node k donated by cluster application ca is borrowed by cluster application ca' . T_s and T'_s respectively denote the request service time of ca and ca' . λ and λ' respectively denote the requests arrival rate of ca and ca' on node k . $\rho_{threshold}$ denotes the CPU utilization upper limit constraint specified by ca in SLA. $r_{threshold}$

denotes the average response time constraint specified by ca . We model a cluster application with multiple M/G/1 queues to capture the dynamics between request arrival rates, resource utilization and average response time. The average response time of ca at each server node before donation can be computed as follows under the M/G/1 queuing model:

$$E[r] = E[T_s] + \frac{\rho E[T_s](1+C_e^2)}{2(1-\rho)} \quad (1)$$

where $E[T_s]$ is the average request service time, C_e is the coefficient of variation of the request service time. The average request service time of ca at each server node is the sum of average service time of each component per request at low request rate (when there is no resource competition or queuing in the system). $E[T_s]$ and ρ can be obtained from application performance profiling. Consider the average response time constraint $E[r] < r_{threshold}$ and formula (1), we have

$$\rho < \frac{2(r_{threshold} - E[T_s])}{E[T_s](C_e^2 - 1) + 2r_{threshold}} \quad (2)$$

Because of the CPU utilization constraint $\rho < \rho_{threshold}$, we get the server resource utilization limit ρ_{limit} of ca .

$$\rho < \rho_{limit} = \min(\rho_{threshold}, \frac{2(r_{threshold} - E[T_s])}{E[T_s](C_e^2 - 1) + 2r_{threshold}}) \quad (3)$$

Inequality (3) shows that as long as we keep the node resource utilization (workload intensity) below ρ_{limit} , both the resource utilization and response time constraints of ca could be met statistically. Further, from Little's Law and (3), we have the requests arrival rate limit λ_{limit} of application ca .

$$\lambda < \lambda_{limit} = \rho_{limit} / E(T_s) \quad (4)$$

Inequality (4) means that as long as the workload arrival rate is kept below λ_{limit} , the server node resource utilization limit will be met. Although inequality (3) and (4) are obtained when ca uses the server node exclusively, the results can be further used to approximate the system behavior when ca and ca' share the server node, because most of ca' workloads will be restrained through admission control when the server resource utilization and request arrival rates approach their limits. To keep ca' from aggressively consuming the borrowed server resources, we must keep its workload arrival rates under control in a manner flexible enough to be adaptive to the changes of the donor application ca 's workload. We accomplish this as follows. When the server is borrowed by ca' , we have $\rho = \lambda \cdot E(T_s) + \lambda' \cdot E(T'_s)$ from Little's Law. With this formula and inequality (3), we have

$$\lambda' < \lambda'_{limit} = \max[(\rho_{limit} - \lambda E(T_s)), 0] / E(T'_s) \quad (5)$$

Inequality (5) indicates that with the increasing workloads of ca , requests of ca' for the shared node will be restrained so that the SLA of the donor can be guaranteed.

3.2 Optimal Server Selection Algorithm

When the cluster center has more than one candidate donated nodes for an overloaded application, how to select the most appropriate nodes becomes another issue. In this section, we design an optimal server selection algorithm called “least nodes matching algorithm” to address this problem.

We call the set of candidate server nodes of a cluster center *resource pool* (abbr. RP), which includes both the donated server nodes and free server nodes. We refer to the difference between an overloaded application’s current capacity and its expected capacity as the overloaded application’s *needed capacity*. $\rho_{\text{limit, } ca_j}$ denotes the server resource utilization limit of the application ca_j . $\rho_{\text{current, } node_k}$ denotes the current resource utilization of $node_k$. We define the *spare capacity* of $node_k$ denoted by $Cap(node_k)$, as follows.

- If $node_k$ is a donated node and belongs to application ca_j , $Cap(node_k) = \rho_{\text{limit, } ca_j} - \rho_{\text{current, } node_k}$.
- If $node_k$ is a free server node, $Cap(node_k)$ is estimated by $\frac{1}{M} \sum_{ca_j \in CA} \rho_{\text{limit, } ca_j}$.

$Cap(node_k)$ models the remaining capacity of a server node which could be safely utilized to handle additional workload. Now we describe how to model the needed capacity for an overloaded application, denoted by $\Delta Cap(ca_j, node_k)$.

- Suppose application ca_j suffers from *local working node overload* (we discuss single-node overload here and treat multi-nodes simultaneous overload as a serial of single-node overload in smaller time scale). We can derive $\Delta Cap(ca_j, node_k) = \tilde{\lambda}_{node_k} E[T_s] - \rho_{\text{limit, } ca_j}$ from Little’s Law and formula (3). $\tilde{\lambda}_{node_k}$ denotes the estimated requests arrival rates when overload happens to $node_k$, and $E[T_s]$ denotes average request service time of ca_j .
- Suppose application ca_j suffers from the *whole application overload*, which is indicated by assigning NULL to $node_k$. ψ_j denotes the number of active sessions of ca_j when overload occurs. $\Delta \psi_j$ denotes the excessive new sessions of ca_j imposed upon cluster load dispatcher. We use the term $|P_j| \rho_{\text{limit, } ca_j} / \psi_j$ to estimate the average capacity demand per session, where $|P_j|$ denotes the number of server nodes of ca_j . Because the precondition of whole application overload is that all resource capacities are fully-utilized, we can safely calculate $\Delta Cap(ca_j, node_k) = \Delta \psi_j (|P_j| \rho_{\text{limit, } ca_j} / \psi_j)$.

Based on the concept of *needed capacity* of an overloaded application, we design “least nodes matching algorithm” for optimal server selection. The main idea is that the number of selected nodes should be least with respect that the needed capacity of the overloaded application can be met if possible. Suppose the resource pool RP is presented with a List structure with server node elements sorted on their capacity in decreasing order. The resource pool supports the following four operations:

- $\text{Add}(node_k)$: insert $node_k$ into RP according to its $\text{Cap}(node_k)$;
- $\text{Remove}(node_k)$: remove $node_k$ from RP;
- $\text{Head}()$: return the node of the largest spare capacity and remove it from RP;
- $\text{Before}()$: find and return the node with spare capacity exactly no less than ,otherwise NULL is returned.

The least nodes matching algorithm is described in Algorithm 1.

Input: an overloaded application ca_j , an overloaded node $node_k$.

Output : a set of candidate nodes that can be borrowed.

```

1.   SelectNodes( $ca_j, node_k$ )
2.   {
3.     Result= ; C= $\Delta\text{Cap}(ca_j, node_k)$ ; K=RP.Before(C);
4.     if ( K!= NULL ) Result={K}; return Result;
5.     if ( K== NULL )
6.       { while(K== NULL && RP!= ) do
7.         { head= RP.Head();
8.           Result= Result  $\cup$  {head};
9.           C = C-Cap(head);
10.          K= RP.Before(C);
11.        }
12.        if ( K!= NULL ) ,Result= Result  $\cup$  {K}; return Result ;
13.        if ( RP== ), return Result ;
14.      }
15.    }

```

Algorithm 1. Least nodes matching algorithm

Based RP nodes sorting, least nodes matching algorithm employ greedy strategy to select the nodes. It's easy to prove that this algorithm achieve an optimal node selection in that number of selected nodes would be least with the needed capacity of the overloaded application exactly met if possible.

4 Threshold-Driven Overload Handling with COROB

With COROB, we have resolved how to optimally selected server nodes for overloaded applications to borrow and how to guarantee the SLA constraints for donor applications during resource borrowing. This section makes use of all these techniques to describe how to conduct cluster application overload handling with COROB through a threshold-driven overload handling method.

4.1 Threshold Definition

We define a set of performance-related thresholds to indicate whether an application is overloaded, when overload handling actions such as resource borrowing should be

taken, and when the borrowed nodes should be returned. The proposed thresholds are sensitive and adaptive enough to model the dynamics of the workloads, as well as resource action time (latency introduced by resource borrowing and application reconfiguration time). The cluster center will use these thresholds definition to judge application performance and to drive system overload handling.

- **Judging cluster application overload**

Most hosted online services use average response time as service quality criterion and specify response time targets in SLAs. With respect to these kinds of cluster applications, response time violation is a straightforward signal for application overload. However, frequently gauging server-side response time will introduce mutex cost from the requests interception, resulting in concurrency loss. We propose a resource utilization monitoring approach, which is less costly, plus low frequently random response time gauging. With the knowledge of request arrival distributions, we can derive resource utilization threshold for an application, excess of which can be used to indicate response time violation with high probability. We refer to this threshold of application ca_j as *crisis utilization threshold*, denoted by $\rho_{T\text{-crisis}, ca_j}$.

Based on results in section 3.2, we have $\rho_{T\text{-crisis}, ca_j} = \rho_{\text{limit}, ca_j}$, where $\rho_{\text{limit}, ca_j}$ is the resource utilization limit for application ca_j . When the utilization of a node from ca_j exceeds $\rho_{T\text{-crisis}, ca_j}$, we conclude that local working node overload happens to ca_j .

When average utilization of all nodes of ca_j exceeds $\rho_{T\text{-crisis}, ca_j}$, we conclude the whole application overload occurs.

- **Modeling resource action time**

There is always latency between overload occurrence and the moment the borrowed nodes can partake the excessive workload. We refer to this latency as *resource action time*. If not carefully treated, overload will be magnified during resource action time. We propose an adaptive utilization threshold $\rho_{T\text{-alarm}}$ called *alarm utilization threshold* by taking resource action time into account. $\rho_{T\text{-alarm}, node_k}$ of application ca_j on $node_k$ can be computed as follows. T denotes average resource action time of ca_j , which can be obtained from cluster center operation statistics. $\Delta\rho_{T, node_k}$ denotes the positive amplitude of resource utilization vibration during interval T . And we have

$$\rho_{T\text{-alarm}, node_k} = \rho_{\text{limit}, ca_j} - \Delta\rho_{T, node_k} \cdot \rho_{\text{current}, node_k} / \rho_{\text{limit}, ca_j} \quad (6)$$

When the utilization of $node_k$ from ca_j exceeds $\rho_{T\text{-alarm}, node_k}$, we conclude that local working node overload likely happens to $node_k$ and overload handling actions can be taken to gain time for resource borrowing and reconfiguration. As node resources are fully utilized (i.e. $\rho_{\text{current}, node_k} \approx \rho_{\text{limit}, ca_j}$), we have $\rho_{T\text{-alarm}, node_k} \approx \rho_{\text{limit}, ca_j} - \Delta\rho_{T, node_k}$, which means alarm utilization threshold exactly gains time for resource actions. On the other extreme, when node resources are mostly idle (i.e. $\rho_{\text{current}, node_k} / \rho_{\text{limit}, ca_j} \rightarrow 0$), we have $\rho_{T\text{-alarm}, node_k} \approx \rho_{\text{limit}, ca_j}$, which means alarm threshold won't waste the server

node resource. To define $\rho_{T\text{-alarm, node}_k}$, cluster center infrastructure needs to monitor and get statistics about $\Delta\rho_{T,\text{node}_k}$ for each nodes every T interval.

- **Returning borrowed resources**

The borrowed nodes should be returned to the cluster center as early as possible. In COROB, there are two occasions to return the nodes. First, when all the sessions of the borrowing applications naturally terminate on a borrowed node, the node will be returned. Secondly, when loads on the borrowed nodes are low and the utilization on the original servers decrease to a low level so as to have enough resources to take over the workload partaken by the borrowed node, we migrates these workload and return the borrowed node. The second situation can only applied for stateless-session applications. We propose the *low utilization threshold* for every borrowed node denoted by $\rho_{T\text{-low, node}_k}$ for node_k from ca_j . We compute low utilization thresholds as

$\rho_{T\text{-low, node}_k} = \rho_{\text{start_to_donate}}$, where $\rho_{\text{start_to_donate}}$ is the utilization threshold specified in donor application ca_j 's SLA to indicate that the node can be borrowed only if the utilization goes below $\rho_{\text{start_to_donate}}$. In the SLA example taken in section 3.1, $\rho_{\text{start_to_donate}} = 20\%$. We define the *low throughput threshold* for borrowing applications, denoted by $\lambda_{T\text{-low}}$. This value can be specified by the cluster center administrator. When (1) request rates of borrowing application ca_l on borrowed node node_k is below $\lambda_{T\text{-low}}$, (2) resource utilization of node_k is below $\rho_{\text{start_to_donate}}$, and (3) the resource utilization of original node node_l from borrowing application ca_l is smaller than the *utilization migration threshold* $\rho_{T\text{-migrate, node}_l} = \min(\rho_{\text{limit, } ca_l} - E[T_s]\lambda_{T\text{-low}}, \rho_{T\text{-alarm, node}_l})$, which indicates the original node has enough resource to take over the ca_l workload on node_k , the cluster center will trigger workload migration and return borrowed node node_k .

All thresholds definitions are summarized in Table 1.

Table 1. Thresholds definition in COROB framework

Threshold	Meaning
$\rho_{T\text{-crisis}}$	<i>crisis utilization threshold</i> , used to indicate whether an application is overloaded.
$\rho_{T\text{-alarm}}$	<i>alarm utilization threshold</i> , used to indicate whether an application is likely to be overloaded, gaining time for resource borrowing and configuration.
$\rho_{T\text{-low}}$	<i>low utilization threshold</i> , used to indicate whether the load of borrowed node low enough to be returned.
$\lambda_{T\text{-low}}$	<i>low throughput threshold</i> , used to indicate whether the workload from borrowing application is weak enough to be migrated.
$\rho_{T\text{-migrate}}$	<i>utilization migration threshold</i> , used to indicate whether the original node has enough resources to take over workload on the borrowed node.

4.2 Threshold-Driven Overload Handling Algorithm

COROB uses a threshold-driven algorithm to conduct cluster application overload handling. The main idea is that the cluster center adaptively triggers overload-related events according to above thresholds definitions, and drives the overload handling procedure (i.e. selecting nodes, borrowing resource, partaking workload and returning nodes). Let's recollect some notations before describing the overload handling algorithm. ca_j denotes a cluster application, and $node_k$ denotes a server node of the cluster center. ρ_{limit} denotes the server resource utilization limit of an application computed by formula (3). λ_{limit} denotes the request arrival rate limit of an application computed by formula (4). All nodes of an application have the same ρ_{limit} and λ_{limit} values. Suppose $node_k$ belongs to ca_j , then we have $\lambda_{\text{limit},node_k} = \lambda_{\text{limit},ca_j}$. T denotes the average resource action time of an application. λ_{current} and $\lambda'_{\text{current}}$ of $node_k$ respectively denote the request arrival rates of the donor application and borrowing application on $node_k$. B_{ca_j} denotes the borrowed nodes set of ca_j , and R_{node_k} denotes the borrowing applications set on $node_k$. The algorithm is described in Algorithm 2.

Input: $\rho_{\text{limit}}, \lambda_{\text{limit}}, \rho_{T-\text{crisis}}, \lambda_{T-\text{low}}, \rho_{\text{start_to_donate}}, T$ for every ca_j ($1 \leq j \leq M$);
 $\rho_{\text{current}}, \lambda_{\text{current}}, \lambda'_{\text{current}}, \rho_{T-\text{low}}$ for every $node_k$ ($1 \leq k \leq N$).

Output: a set of parameters controlling overload handling.

1. **for** $k=1$ to N **do** $R_{node_k} = \emptyset$; **for** $j=1$ to M **do** $B_{ca_j} = \emptyset$;
2. **while**(true) **do**{
3. **for** $k=1$ to N **do** { // handle each node
4. Compute $\rho_{T-\text{alarm},node_k}$ according to formula (6) every interval T_{ca_j} ;
5. **if** ($R_{node_k} \neq \emptyset$) { // assuming $node_k$ belongs to ca_j and borrowed by ca_j
6. $\lambda'_{\text{limit},node_k} = \max[(\rho_{\text{limit},ca_j} - \lambda_{\text{current},node_k} E(T_{s,ca_j}), 0)] / E(T'_{s,ca_j})$ (see section 3.1);
7. }
8. Assign $\lambda_{\text{limit},node_k}, \lambda'_{\text{limit},node_k}$ to cluster center, and perform admission control;}

Line 2~26 mainly accomplish local working node overload handling, borrowed nodes returning and performance guarantee for donor applications during resource borrowing. The cluster center checks the cluster nodes performance one by one according to related thresholds, and drive resource borrowing to react to overload occurrence. Line 4~6 computes the threshold $\rho_{T-\text{alarm}}$ and admission control rate λ'_{limit} for each node, if the node is borrowed. Line8~9 update admission control parameter to reflect the donor application workload change, and perform requests admission control for each node. Line 10~17 handle the local overload by borrowing nodes from the cluster center and expand the overloaded application components to the borrowed

```

9  if ( $\rho_{\text{current, node}_k} > \rho_{T\text{-crisis, } ca_j}$ ) {  $\lambda'_{\text{limit, node}_k} = 0$ ;  $\lambda_{\text{limit, node}_k} = 2/3 \lambda_{\text{limit, node}_k}$ ; } continue;
10. if ( $\rho_{\text{current, node}_k} > \rho_{T\text{-alarm, } node_k}$ ) { // assuming  $node_k$  belongs to  $ca_j$ 
11.   BorrowedNodes = RP.SelectNodes( $ca_j$ ,  $node_k$ );
12.   for  $n \in BorrowedNodes$  and  $n \notin P_j$  do{
13.      $ca_j$ .Expand( $Config_j(node_k)$ ,  $n$ );
14.      $R_n = R_n \cup \{ca_j\}$ ;
15.   }
16.    $B_{ca_j} = B_{ca_j} \cup BorrowedNodes$  ;
17. }
18. if ( $\rho_{\text{current, node}_k} < \rho_{\text{start\_to\_donate, } ca_j}$ ) { // assuming  $node_k$  belongs to  $ca_j$ 
19.   if( $R_{node_k} == \emptyset$ ) RP.Add( $node_k$ ); continue;
20.   if ( $node_k \in B_{ca_l}$  and  $\lambda'_{\text{current, node}_k} < \lambda_{T\text{-low, } ca_l}$  and  $\rho_{\text{current, node}_l} <$ 
         $\min(\rho_{\text{limit, } ca_l} - E[T_{s, ca_l}] \lambda_{T\text{-low, } ca_l}, \rho_{T\text{-alarm, } node_l})$ ) {
21.     //  $node_k$  is borrowed by  $ca_l$ , and  $node_l$  is the original overloaded node
22.      $ca_l$ .Migrate( $Config_l(node_k)$ ,  $node_k$ ,  $node_l$ );
23.      $B_{ca_l} = B_{ca_l} - \{node_k\}$ ;  $R_{node_k} = R_{node_k} - \{ca_l\}$ ;
24.   }
25. }
26. } // end of handle each node
27. for  $j=1$  to  $N$  do { // handle each application
28.   if (average resource utilization of  $ca_j > \rho_{T\text{-crisis, } ca_j}$ ){
29.     BorrowedNodes = RP.SelectNodes( $ca_j$ , NULL);
30.     for  $n \in BorrowedNodes$  and  $n \notin P_j$  do{
31.        $ca_j$ .Expand( $Components_j$ ,  $n$ );
32.        $R_n = R_n \cup \{ca_j\}$ ;
33.     }
34.      $B_{ca_j} = B_{ca_j} \cup BorrowedNodes$  ;
35.   }
36. } // end of handle each application
37. } //end of algorithm

```

Algorithm 2. Threshold-driven overload handling algorithm

one to actualize resource borrowing. Line 19 helps donate an under-utilized node to cluster center. Line 20~24 take back an under-utilized borrowed node by migrating

the components and request loads back to the original server. Line 27~36 handle the whole application overload for each application.

With the help of COROB, as well as adaptively computed thresholds, this algorithm not only can resolve local node and the whole application overload, but share the underutilized server resources in a controlled manner.

5 Experiments and Results

In this section, some preliminary experiments are presented to demonstrate the efficacy of COROB for a middleware-based service hosting cluster prototype. The results validate the threshold-driven overload handling algorithm for performance guarantee under partial application overload conditions.

5.1 Experimental Setup

Service hosting cluster environment testbed. We construct a prototype service hosting cluster environment as a testbed for resource borrowing framework validation, where OMG CORBA 2.5 standard-compliant objects can be hosted as network services. The cluster environment is built on StarBus+ [5], which is a comprehensive CORBA-complaint middleware suite with the features such as object request broker supporting multi-* quality of service, component model, and integration with Web Service. The hosting cluster environment consists of IIOP session scheduler, a resource management controller, working node machines and corresponding node managers, as depicted in Fig 2. The heart of the cluster environment is the resource management controller, called *COROB controller*, which takes charges of initial working nodes allocation, SLA monitoring and dynamic resource borrowing for overloaded hosted applications. All hosted applications' contracted SLAs are stored in COROB controller, and variety of performance statistics, such as CPU utilization, average response time and throughputs, are periodically pushed to COROB controller from applications' working node managers. These raw data are used to compute the set of thresholds values, such as crisis utilization thresholds, which are further employed by COROB controller to judge whether the SLA of each application were violated. When overload events are triggered, the controller responds with selecting and borrowing under-utilized working node from donor applications. The COROB controller will tell the scheduler to dispatch excessive workload to newly borrowed resource nodes, after corresponding node managers are notified to dynamically load overload applications components onto the borrowed nodes.

In experiments, the service hosting cluster environment is concretely made up of five machines with 2.0 GHz Pentium IV processors and 512 MB RAM connected by 100 Mbps Ethernet. All machines run Windows 2000 Server with SP 2 and StarBus+ runtime. One machine is used to host COROB controller process and the cluster scheduler, while the others serve as hosting working nodes. Each working node run a hosting container sever, where Windows DLL-based StarBus+ applications can be dynamically loaded and hosted.

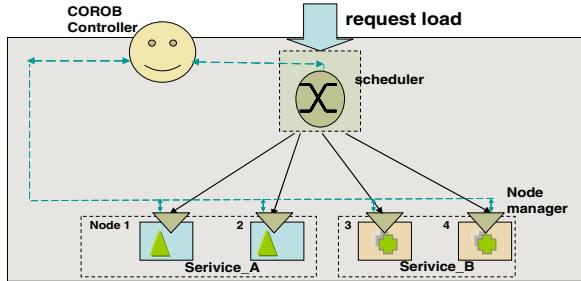


Fig. 2. Prototype service hosting cluster structure used in the experiments

Experiments design and configurations. The cluster environment hosts two applications, Service_A and Service_B, both of which are CPU-intensive applications and use the method transaction_A and transaction_B respectively as the service interface entry. Initially, both applications are allocated two machines under some contract (node 1 and 2 for Service_A, and node 3 and 4 for Service_B, see Fig 2). Both applications require response time be kept within 150 ms, which is the primary QoS constraints in their SLAs. And Service_B declares to be a resource donor in its SLA and has following resource donation constraints: keeping CPU utilization below 90% and average response time less than 150 ms during resource sharing. Note that for computation simplicity, both transaction_A and transaction_B methods are designed to fulfill constant computation, which means Service_A and Service_B have constant service time. LoadRunner 7.8 [6] is employed to generate virtual request sessions from a separate machine, and is used to collect experiments statistics, such as end-2-end response time and resource utilization etc. In our experimental scenario, 40 users sessions are gradually activated for Service_A, and Service_B is fed with 2 sessions. Each user iterates calling corresponding service methods, with 500 ms think time. The scheduler balances workloads among working nodes for each application by default.

5.2 Resource Borrowing and Overload Handling

To demonstrate the efficacy of resource borrowing framework, we conduct two experiments with and without COROB support. First, we conduct an experiment without resource borrowing mechanism supports between applications. Fig 3 and Fig 4 present the experimental statistics obtained by LoadRunner. Fig 3 shows that the average response time of Service_A's method transaction_A begins violating average response time constraint at about 01:53, with severe performance degradation following. Fig 4 plots the CPU utilization of four working nodes, from which we see Service_B's working nodes are mostly under-utilized when Service_A incurs overload.

In the second experiment, both Service_A and Service_B face the same workload as prior, while COROB is started to facilitate resource borrowing and overload handling. Fig 5 shows the resulting performance statistics of both applications through LoadRunner monitoring facility. We see the average response time of both applications

are well kept under 150 ms most of the time. At the same time, the CPU utilization of borrowed nodes (node 3 and node 4 in order) is kept under contracted 90%.

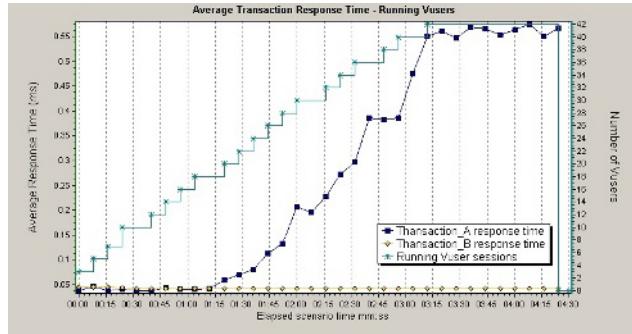


Fig. 3. Average transaction response time vs. Running sessions without COROB

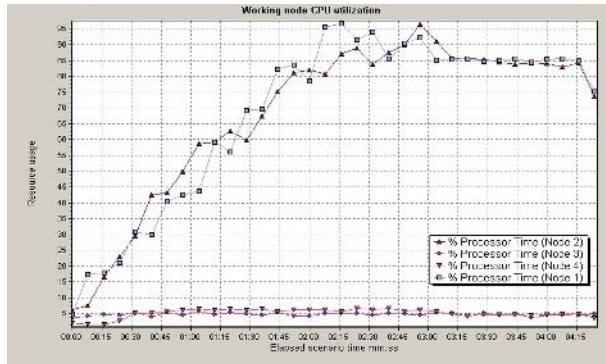


Fig. 4. Working node CPU resource utilization without COROB support

Now we detail how COROB helps overload handling through threshold-driven overload handling algorithm. The basis of the algorithm is the thresholds computation, and we take $\rho_{T\text{-crisis}}$ and $\rho_{T\text{-alarm}}$ computation as example to illuminate this process.

- **Computation of $\rho_{T\text{-crisis}}$ for Service_A.** As $\rho_{T\text{-crisis}, \text{Service}_A} = \rho_{\text{limit}, \text{Service}_A}$, we need to compute $\rho_{\text{limit}, \text{Service}_A}$ first. According to Service_A configuration in this experiment, we have $\rho_{\text{threshold}, \text{Service}_A} = 90\%$, $r_{\text{threshold}, \text{Service}_A} = 150\text{ms}$. From Little's Law $\rho = \lambda \cdot E(T_s)$ and experimental profiling, we have $E(T_s) = 35\text{ ms}$. As Service_A has constant service time, the coefficient of variation of service time $C_e = 0$. Following formula (3), we have $\rho_{\text{limit}, \text{Service}_A} = 87\%$, which means $\rho_{T\text{-crisis}, \text{Service}_A}$ equals to 87%.

- **Computation of $\rho_{T\text{-alarm}}$ for Service_A.** First, through profiling we monitor that resource action time of Service_A is less than 100 ms, i.e. $T=0.1$ second. During the experiment, we gauge that the positive amplitude of resource utilization vibration of node 1 and node2 is less than 2% during 0.1 second interval (see Fig 4). So we have $\rho_{T\text{-alarm}, \text{Service}_A} = 87\% - 2\% = 85\%$, when these server nodes utilization approaches $\rho_{\text{limit}, \text{Service}_A}$.

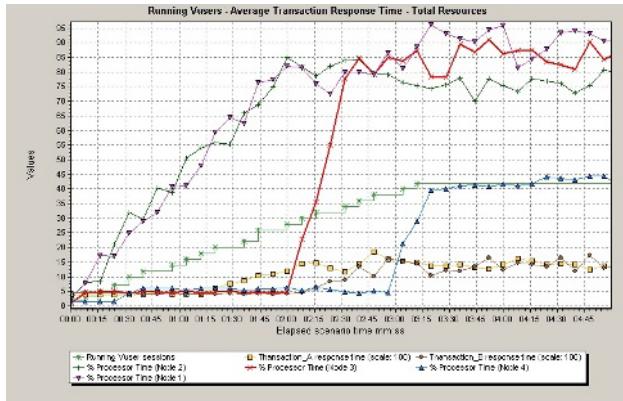


Fig. 5. Overload handling and response time guarantee with COROB

With $\rho_{T\text{-crisis}}$ and $\rho_{T\text{-alarm}}$ computed for each hosted applications, COROB can use these thresholds to drive resource borrowing in response to application overload. In Fig 4, we see node 2 of Serviec_A first reaches 85% CPU utilization at around 02:00, which triggers an overload event followed by resource borrowing process. The COROB controller uses the nodes matching algorithm described in section 3.3 to conduct server selections. As the overload is caused by increasing sessions on the scheduler, COROB believes this is the whole application overload. From Service_B, node 3 is borrowed to partake the increasing sessions. At around 02:55, node 3 reaches 85% CPU utilization, which makes Serviec_A continue to borrow node 4 to relieve overload.

6 Related Work

The importance of providing performance guarantee and resource management for service hosting centers has been recognized in Grid and utility computing communities. Previous work on resource management and overload control in Internet platforms spans several areas. We briefly review prior work that is most relevant to COROB framework in two aspects.

Resource management in shared cluster. Condor [7] is one of the leading projects addressing resource management and scheduling for compute-intensive batch jobs in

Grid environment. It pools together resources belonging to multiple domains and provides mechanisms for job queuing, scheduling, resource monitoring. Océano [13], as a “computing utility” infrastructure for multi-customer hosting on a server farm, supports server-granularity dynamically resource allocation between customers to smoothes out peaks. Océano monitors and reacts to the SLA violation by servers reconfiguration. However, both Condor and Océano operate at a relatively coarse granularity, and don’t base resource allocation decisions on theoretical performance analysis. Chandra [4] quantifies the effects of different resource granularities on multiplexing benefits in a hosting data center, and shows that fine-grained multiplexing will achieve better resource utilization. OnCall[8] employs an economic market-based approach to handle workload spikes by allowing applications to trade server nodes on a free market. Quartermaster[15], as capacity manager service for managing resource pools of enterprise computing environment, implements a trace-based technique that models workload resource demands, their corresponding resource allocations, and resource access quality of service. However, we employ queuing-based analysis approach to estimate both the spare capacities of the donated servers and the capacity requirement of overloaded applications.

Overload management for Internet services. There is a substantial literature regarding admission control, resource reservation, and scheduling in support of Internet services. The feedback control method based on control theory gains more popularity in recent years for web system overload management [9,10,11]. N. Gandhi [11] presents a connections-and-timeout controller to achieve stable CPU utilization of Apache web server. Welsh and Culler [3] propose an overload management solution for Internet services built using the SEDA architecture. Welsh argues that overload management should be considered in the early stage of system design. E. Lassettre [12] proposes an overload protection mechanism to guard Websphere servers cluster by dynamically provisioning servers for an overloaded application from free server pool.

Compared with previous server-granularity-based resource management, this paper studies the feasibility and methods of fine-grained resource sharing and shows how to rationally utilize leftover resources to handle overload in component application hosting environment. We provide rigorous analysis of the effects of resource borrowing, based on which SLAs of donor application can be guaranteed during resource sharing. We advocate an open SLA-based resource donation strategy, and the proposed threshold-driven overload handling algorithm is sensitive and smart enough to capture the dynamics of changing workloads. Our work complements existing admission control techniques by enlarging resource supplies to handle overload.

7 Conclusions

In this paper, we present COROB, a comprehensive fine-grained resource borrowing framework for cluster-based hosting center running multiple component applications. Through COROB, under-utilized server resources can be shared among applications in a controlled way, with SLAs of donor applications guaranteed. Further work will

look at security and robustness of hosted applications, in particular dealing with rogue applications with sound mechanism design.

References

1. W. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the Self-Similar Nature of Ethernet Traffic (extended version). IEEE/ACM Transactions on Network v2, February 1994.
2. A. Andrzejak, M. Arlitt, and J. Rolia. Bounding the Resource Savings of Utility Computing Models. Technical Report HPL-2002-339, HP Labs, Dec. 2002.
3. Matt Welsh and David Culler, Overload Management as a Fundamental Service Design Primitive. In Proceedings of the Tenth ACM SIGOPS European Workshop, Saint-Emilion, France, September, 2002.
4. Abhishek Chandra, Pawan Goyal and Prashant Shenoy, Quantifying the Benefits of Resource Multiplexing in On-Demand Data Centers, Proceedings of the First ACM Workshop on Algorithms and Architectures for Self-Managing Systems (Self-Manage 2003), San Diego, CA, June 2003.
5. Wang H.M., Wang Y.F. and Tang Y.B., StarBus+: Distributed object middleware practice for Internet computing, Journal of Computer Science and Technology, Vol.20, No.4, 2005, pp.542-551.
6. LoadRunner. <http://www.mercuryinteractive.com/>.
7. M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In Proceedings of the 8th International Conference of Distributed Computing Systems, pages 104–111, June 1988.
8. James Norris, Keith Coleman, Armando Fox, George Candea , OnCall: Defeating Spikes with a Free-Market Application Cluster ,IEEE International Conference on Autonomic Computing (ICAC'04) 2004
9. T.F. Abdelzaher, et. al., "Feedback Performance Control in Software Services," IEEE Control Systems, 23(3), June 2003.
10. T. Abdelzaher, K. G. Shin, and N. Bhatti. Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach. IEEE Transactions on Parallel and Distributed Systems, 13(1), Jan. 2002.
11. Y. Diao, N. Gandhi, J.L. Hellerstein, S. Parekh, and D.M. Tilbury, "MIMO control of an Apache Web server: Modeling and controller design," in Proc. American Control Conf., Anchorage, AK, May 2002, pp. 4922-4927.
12. E. Lassettre, D. W. Coleman, Y. Diao, S. Froehlich, J. L. Hellerstein, L. Hsiung, T. Mumment, M. Raghavachari, G. Parker, L. Russell, M. Surendra, V. Tseng, N. Wadia, and P. Ye, "Dynamic Surge Protection: An Approach to Handling Unexpected Workload Surges With Resource Actions That Have Dead Times," IFIP/IEEE Workshop on Distributed Systems: Operations and Management, October 2003.
13. K. Appleby, S. Fakhouri, L. Fong, M. K. G. Goldszmidt, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger. Océano - SLA-based Management of a Computing Utility. In Proceedings of the IFIP/IEEE Symposium on Integrated Network Management, May 2001.
14. Yufeng Wang, Huaimin Wang, Yan Jia, Dianxi Shi, Bixin Liu, A SLA-based Resource Donation Mechanism for Service Hosting Utility Center, in Proceedings of The 4th International Conference on Grid and Cooperative Computing (GCC 2005), Beijing, China, November 30-December 3, 2005. (accepted)
15. Jerry Rolia, Ludmila Cherkasova, Martin Arlitt, Artur Andrzejak, A Capacity Management Service for Resource Pools, In Proceedings of the Fifth International Workshop on Software and Performance (WOSP 2005), ACM, July 2005.

Accessing X Applications over the World-Wide Web

Arno Puder and Siddharth Desai

San Francisco State University,
Computer Science Department,
1600 Holloway Avenue,
San Francisco, CA 94132
{arno, sgd1977}@sfsu.edu

Abstract. The X Protocol, an asynchronous network protocol, was developed at MIT amid the need to provide a network transparent graphical user interface primarily for the UNIX Operating System. Current examples of Open Source implementations of the X server, require specific software to be downloaded and installed on the end-user's workstation. To avoid this and other issues involved in the conventional X setup, this paper proposes a new solution by defining a protocol bridge that translates the conventional X Protocol to an HTTP-based one. This approach makes an X application accessible from any web browser. With the goal of leveraging the enormous browser install base, the web-based X server supports multiple web browsers and has been tested to support a number of X clients.

1 Motivation

The staggering rate at which the World-Wide Web has grown over the last decade is evidenced by the number of websites that are accessible over the Internet today. Web browsers, the end-user applications that connect to these websites and display content have also evolved at an impressive rate. Originally based on a document-centric architecture, where a web browser would only display static HTML pages, much work has been done to define extensions that allow for operational interactions. Thus web browsers have generic interfaces for web-based applications.

Although the most popular browsers are freely available and readily downloadable from the Internet, most operating systems today bundle a web browser. As a result, the install base of web browsers has dramatically increased. This has been leveraged by companies that do business online and software providers who have migrated their native client interfaces and made them web-based. The obvious technical benefit is that with all prerequisite software already installed, installation and configuration for such web-based applications is dramatically reduced or even eliminated altogether.

In this paper we introduce a technique that allows to access X applications – based on MIT's X Protocol – through a web browser. This will make X applications that traditionally require an X server to be installed, accessible in a web

browser without having to modify and rebuild those applications. Our proposal is based on a protocol bridge, that translates the X Protocol to an HTTP-based one.

This paper is organized as follows: in Section 2 we give a brief introduction to generic clients. Section 3 will discuss various design alternatives. Section 4 introduces the architecture of our protocol bridge we call XWeb. Section 5 describes our prototype implementation of XWeb as well as some performance measurements. Section 6 discusses related work, and in Section 7 we provide a conclusion and outlook.

2 Generic Clients

In this section we introduce the notion of a *generic client*. A generic client is controlled by an end-user to access a remote application. We use the term generic to denote the fact that the client is not specialized for any particular application, but rather provides access to *a priori* unknown applications. Two examples of generic clients that we will discuss here are X servers and web browsers.

By being a network-aware graphical user interface system, X allows the separation of an application's processing and its output (see [8]). In X Windows, the application is called the X client, whereas the output is rendered at the X server. The X server is therefore located at the side of the end-user (see Figure 1).

The X client and X server communicate with each other via the X Protocol. The X Protocol is an asynchronous, binary protocol. The X client sends the window content to be rendered to the X server, whereas the X server sends user input (e.g., keystrokes or mouse movement) to the X client. Interestingly, the X Protocol does not support widgets such as buttons, listboxes, or radio buttons. In X, everything is rendered as a graphical image. If the user interface of an X client requires a button, it needs to be manually drawn by the X client. This accounts for the fact that many different X applications have a different look-and-feel, since every application can draw its user interface widgets in a different way.

Because the X Protocol is about images and not widgets, another implication is that there is a fair amount of network traffic between an X client and the X server. E.g., every keystroke of the user requires a roundtrip communication with

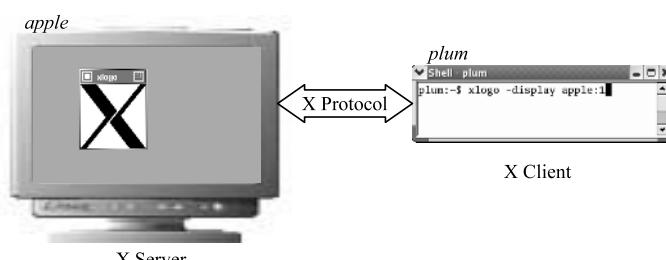


Fig. 1. X Window Protocol

Table 1. Comparing X servers and web browsers

	X server	Web browser
Granularity	Images	Widgets
Execution platform	None	Java, JavaScript
Communication	Low latency	High latency
Protocol	X Protocol (asynchronous, binary)	HTTP (synchronous)
Platforms	WeirdX, Cygwin/X, HummingBird Exceed	Mozilla, Firefox, IE

the remote application. The consequence is that the X Protocol works best in low-latency, high-bandwidth networks.

Another example of a generic client is a web browser. Initially meant to render static documents only, the World-Wide Web (WWW) has evolved to support operational interactions. User interfaces are described through HTML (Hyper-Text Markup Language) that are downloaded via HTTP (Hyper-Text Transport Protocol) from a remote web server. HTML allows the description of feature-rich user interfaces supporting widgets such as buttons and listboxes. Since a web browser supports the most common widgets natively, the look-and-feel of web applications is more homogeneous than that of X applications.

Table 1 summarizes the main differences between web browsers and X servers. Whereas an X server effectively only render images, web browsers support complex widgets. Web browsers also feature an execution platform based on Java and JavaScript that allows application-specific code to run on the client side. The X Protocol is an asynchronous protocol, by which we mean that both X client and X server can send Protocol Data Units (PDUs) independent of each other. On the other hand, HTTP is a synchronous protocol where the web browser determines when to interact with the remote web server. The web server cannot send a PDU independently to the web browser.

The goal of this paper is to introduce an architecture that allows access to X clients from within a web browser. While no one disputes the ubiquity of web browsers that would give almost universal access to X applications, we need to motivate one important assumption: that there are sufficiently many X applications that make this endeavor feasible. Clearly, the well-known X applications such as xclock, xedit, or xcalc would hardly justify our motivation. Upon closer inspection, there are many GUI-libraries that support X Windows. There are several C/C++ based libraries with an X Protocol binding, such as Qt or GTK. On the Java side, Sun Microsystems offers X-based implementations of their AWT and Swing toolkits. Effectively, every Java application with a GUI is also an X application.

3 Design Alternatives

There are several different design alternatives on how to provide universal access to X applications. Before we discuss the various alternatives, we first want to explicitly state the design goals we expect of a general solution:

1. Provides ubiquitous access to any legacy X application.
2. Runs in all major web browsers.
3. Does not require the installation of additional software.
4. Accessible to non-tech savvy users.

Based on these design goals, we discuss various options in designing a possible solution along with their advantages and disadvantages. In the following sections, we discuss the options of installing a local X server, running an X server as an applet, and a HTTP-based solution in detail.

3.1 Local X Server

The most obvious solution would be to install a local X server. Several products exist – both commercial and as Open Source – that allow to run an X server on all major platforms including Windows, such as WeirdX, Cygwin/X, or HummingBird Exceed (see [5], [12], [4]). In the most straightforward configuration, the X server is displayed in its own window on the hosting windowing system. However, there are two problems related to this solution: the end-users unwillingness to install additional software as well as firewall issues.

Given a choice, end-users either intentionally or unintentionally choose not to install additional software if they have an already existing solution and the benefits of the alternative are not immediately apparent. As a specific example of the reluctance of end-users to explicitly install software can be seen by the proportion of Windows users who use Internet Explorer, despite security issues compared to other browsers. Internet Explorer currently owns more than 85% of the market share (see [13]), primarily because it is bundled along with Windows. If end-users are reluctant or simply do not bother to use easy-to-install software such as alternate web browsers, they would not be willing to install an X server.

Another issue is related to firewalls that may prevent the X client and X server from communicating with each other, if both are on opposite sides of a firewall. Generally, firewalls block most ports to prevent intruders from accessing and compromising services being provided inside the corporate environment. A few ports that provide essential or popular services (such as HTTP) may be left open. However, the ports over which the X Protocol runs are usually blocked. Also note that the X Protocol essentially reverses the client and server relationship: the X client is establishing a connection to the X server. Since end-users also usually run a firewall on their desktop or DSL-modem, this places an additional burden on running a local X server.

3.2 Applet Based Solution

Another solution is to implement the X server as a Java applet. This way, the X server can automatically be downloaded into the browser. Once downloaded, the applet functions as an X server that uses the browser's window to render the output of X clients. WeirdX is an Open Source X server that is based on this idea (see [5]). While this approach solves the problem of a users unwillingness

to install additional software on his or her local machine, it has some issues of its own.

First of all, firewall issues are not resolved as the applet still acts as a server and would have to communicate in the same manner as a locally installed X server would. In addition, the browser's Virtual Machine (JVM) may prevent the applet from opening ports to listen for connections from X clients, as the JVM would deem this to be a security risk. This can be resolved by configuring some of the JVM parameters but once again, the average end-user would find it unappealing.

Another problem is that the future of applets is unclear with Microsoft phasing out their support for its built-in (and outdated) JVM in Internet Explorer in 2007 (see [7]). End-users would either have to download Java Runtime (JRE) software from Sun Microsystems or use a machine where the JRE is pre-installed. Having to deal with JRE installation and patch updates in addition to the browser's own updates would be an added responsibility for the end-user.

3.3 HTTP-Based Solution

A solution in which the web browser can view and interact with X clients over HTTP can solve all of the aforementioned issues. The end-user does not need to download any software and does not have to deal with installation or configuration hassles. She can use the existing web browser on her machine or any other machine she uses. Firewalls are usually configured to allow HTTP. Moreover, unlike X servers which are also TCP servers, web browsers are TCP clients. As a result, the client-server relationship is inverted and this essentially resolves the security issues and no additional security configuration is required. Since the end-user is not installing any software, there are no installation restrictions. The solution is not applet based and does not have dependencies on the JVM plug-in. Naturally, this solution introduces some technical challenges, e.g., how to break the symmetry of HTTP in order to support the asynchronous X Protocol. The rest of this paper describes this solution in detail.

4 Architecture

Our solution follows the HTTP-based solution as outlined in the previous section. Figure 2 gives an high-level overview of the architecture. The main component is a protocol bridge we call the *XWeb Broker*. Information that is sent between the X server and X client over the X Protocol is transformed and sent over HTTP to the web browser by the XWeb broker. Thus, from the X client's point of view, the XWeb broker acts like an X server. Since the X client uses the regular X Protocol to communicate with the XWeb broker, the X client can be used without requiring any modifications to its implementation.

The XWeb broker as shown in Figure 2 acts as a so-called headless X server. This means that while the XWeb broker behaves like an X server, it itself does not render any output and therefore does not require a graphic workstation to

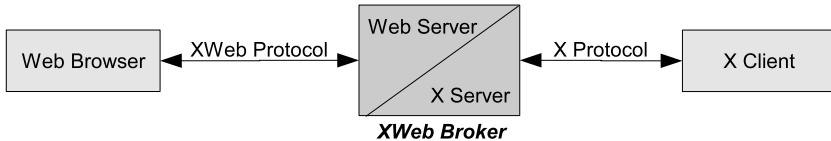


Fig. 2. XWeb overview

run. Instead, the incoming X traffic is transformed and forwarded to the web browser. The web browser is where the visual user interface is rendered. The web browser uses the XWeb protocol to communicate to the XWeb broker. Of course it is not obvious how the XWeb broker can forward information to the web browser, since HTTP only allows the web browser to initiate a request. Our solution to this problem will be discussed in a subsequent section, but first we introduce the informational model.

4.1 Informational Model

The purpose of the informational model is to define a data model that is needed to capture all relevant information for the scope of the XWeb protocol. The informational model of the XWeb protocol is an abstraction of the informational model of the X Protocol. At the core of the X Protocol are *windows* and *panels*. Every X application is contained in its own window. A window can contain multiple panels that form a hierarchy and that are positioned relative to their parent. In Figure 3 the panel ID_3 is a child of panel ID_2 which is itself a child of a top-level window. A panel usually represents a widget such as a button where the X client draws the user interface element in its look-and-feel.

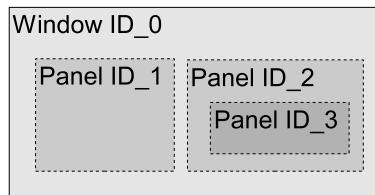


Fig. 3. Window abstraction

The X Protocol offers fine grained drawing operations for lines, rectangles, circles, and other geometric elements. Each of those drawing operations occurs in a panel. Since the X Protocol does not support complex widgets, these have to be drawn manually inside a panel. One option of the XWeb protocol would have been to map the X Protocol elements to equivalent drawing instructions for the web browser. This would result in a one-to-one mapping of X-PDUs to XWeb PDUs. However, this would place a high burden on the client-side protocol

engine of the XWeb protocol, since it would need to understand all the different X-PDUs.

In order to keep the client-side protocol engine as simple as possible (and therefore increase its portability) we limit the informational model to images that will be rendered at the position of their respective panels. This means that the rendering of a panel's content happens inside the XWeb broker, while the web browser only has to load and display the final rendered image. E.g., the X application xlogo draws the X logo using a series of draw instructions. The result will be an image that is captured by the XWeb broker as a PNG (Portable Network Graphics) image.

The informational model also has to include the events that are relevant to the X Protocol. These include mouse events (e.g., mouse move, mouse entered or left a panel, mouse clicked) and key events (e.g., key pressed or released). The informational model has to be mapped to HTML because this is what the web browser is capable of rendering. Windows and panels are mapped to `<div>` elements which serve as boxing elements in HTML. The hierarchy of panels shown in Figure 3 can be translated to the following HTML:

```

1 <!-- HTML -->
2 <div id="ID_0">
3   <div id="ID_1" style="position: absolute; top: 10px;left: 5px"
4     onMouseOver="..." onKeyPressed="...">
5     
6   </div>
7   <div id="ID_2" style="...">
8     
9     <div id="ID_3" style="...">
10    
11  </div>
12 </div>
13 </div>

```

Every `<div>` element has its own ID as well as further attributes that characterize it. The image associated with a panel is simply referenced with the `` tag from HTML. The web browser will automatically load the appropriate image and render it inside the `<div>` box. We use CSS (Cascading Style Sheets) to position the various `<div>` elements relative to their parent. Note that despite its name the “`position: absolute`” argument of the `style`-attribute positions an HTML element relative to its parent. The various events of interests (mouse- and key-events) can be intercepted by registering event-handlers that will invoke appropriate JavaScript functions when they occur.

The HTML shown above only serves as an example on how the informational model is mapped to HTML inside the web browser. Unlike the document-centric usage of HTML where the web browser downloads a complete page and renders it, in XWeb we update the content of the page on a fine-grained basis without

loading a complete HTML-page. This can readily be accomplished manipulating the DOM (Document Object Model) representation of the HTML-page using JavaScript. The following JavaScript excerpt shows how a new panel can be created as a child of panel `ID_3`. The various attributes such as CSS-style or event-handlers can be set accordingly using the DOM-API.

```

1 // JavaScript
2 var new_node = document.createElement("div");
3 var parent_node = document.getElementById("ID_3");
4 parent_node.appendChild(new_node);

```

4.2 XWeb Protocol Data Units

The XWeb broker acts as a protocol bridge. While it uses the regular X Protocol to communicate with X applications, we have devised a new protocol for the communication between the XWeb broker and the web browser. We use HTTP as a transport mechanism for the XWeb Protocol Data Units (PDU). The next section will explain how we achieve an asynchronous protocol on top of HTTP, whereas in this section we focus on the structure of the PDUs.

The web browser needs to be able to marshal and unmarshal a PDU in an efficient way. For that reason the XWeb PDUs are based on XML, since all the major web browsers support XML-parsers. Another advantage is that XML-based PDUs can readily be transmitted via HTTP. The following XML shows an XWeb PDU for creating the panel hierarchy shown in Figure 3. This PDU would be sent from the XWeb broker to the web browser:

```

1 <xweb>
2   <create id="ID_0" type="window" />
3   <create id="ID_1" pid="ID_0" type="panel" />
4   <create id="ID_2" pid="ID_0" type="panel" />
5   <create id="ID_3" pid="ID_2" type="panel" />
6 </xweb>

```

An XWeb PDU is an XML-document whose root element is `<xweb>`. This root element can have one or more children. Each of those child elements could have been transported in their own PDU; marshalling several elements into one PDU makes the XWeb protocol more efficient. The client-side protocol engine processes the child elements from top to bottom. In the example shown above, the four child-elements create one window and three panels. The relationship of IDs (marked by the XML-attribute `id`) to a parent-ID (denoted through the XML-attribute `pid`) determines the nesting of the various panels. Note that up to this point only the hierarchy of the panels has been determined, but not any of their physical attributes such as size and position. The following XWeb PDU demonstrates how these attributes can be defined for panel `ID_1`:

```

1 <xweb>
2   <update id="ID_1">
3     <property name="left" value="10" />
4     <property name="top" value="10" />
5     <property name="width" value="50" />
6     <property name="height" value="50" />
7   </update>
8 </xweb>

```

The XML-tag `<update>` allows to alter various attributes of a panel. In the example shown above, the size and position of the panel are set via the `<property>` tag. The reason for using the attributes `name` and `value` (compared to a more compact form such as `<property left="10"/>`) is that new properties can easily be added without requiring to change the schema of an XWeb-PDU. This effectively makes the protocol as well as the protocol engines more robust. Properties can be changed individually at any point in time after the panel has been created. For most applications the size and position will be only set once and then not changed during the lifetime of the application. Defining the geometry of a panel does not say anything about its content. This is done via another property as shown in the following XWeb PDU:

```

1 <xweb>
2   <update id="ID_1">
3     <property name="image" value="5.png" />
4   </update>
5 </xweb>

```

The property contains a reference to an image with the name `5.png`. Upon receiving the this PDU, the client-side XWeb protocol engine will create the following HTML: ``. Upon creating this HTML-tag, the browser will then automatically issue another HTTP request to load the image `/image/5.png`. Images are therefore loaded asynchronously which requires the XWeb broker to cache those images. Whenever the content of a panel changes, a new image is created and sent via an update-PDU to the web browser.

The PDUs discussed so far are sent from the XWeb broker to the web browser. Whenever an event arises at the web browser, a PDU has to be sent to the XWeb broker to notify it of the event. With respect to the X Protocol, events of interest are keystrokes and mouse movement. The XWeb protocol features the XML-tag `<event>` to describe events. The following XWeb PDU is sent by the web browser to notify the XWeb broker that the mouse has entered the panel with the ID `ID_1` at coordinate (0, 10) and while the mouse was inside this panel, the user pressed the key “H”:

```

1 <xweb>
2   <event id="ID_1" type="mouseEntered" x="0" y="10"/>
3   <event id="ID_1" type="keyPressed" key="H" />
4 </xweb>

```

Note that some PDUs of the X Protocol have no corresponding PDU in the XWeb protocol. One example is the window-expose-event that is sent every time a part of a window becomes visible. This is necessary in the X Protocol, because the X server does not cache window content. Since web browsers do cache the content of a window, this does not pose a problem and therefore does not require a notification to the XWeb broker — the XWeb protocol has no corresponding PDU. Table 2 gives a summary of all XWeb protocol elements.

Table 2. XWeb PDUs

PDU	Description
<xweb>	Toplevel XML-tag for every PDU.
<create>	Creates a window or panel.
<update>	Updates one or more properties of a panel.
<delete>	Deletes a window or panel.
<event>	Sends an event to the XWeb broker.
<property>	Sets one property of a panel.

4.3 XWeb Protocol

Figure 4 gives an overview of the XWeb protocol. The interaction begins when the user visits the URL of the XWeb broker (1). Encoded in the URL is also the X application that the user wishes to launch. Upon receiving the initial request, the XWeb broker starts the X application as a separate process (2) and responds to the web browser with the client-side implementation of the XWeb protocol (3). To achieve browser independence, this implementation is based on JavaScript.

After the client-side of the XWeb protocol has been downloaded into the user's web browser, the protocol engine begins pulling updates (4). Upon receiving the request, the XWeb broker defers the reply until there is something to report back to the browser. Meanwhile, the X application will begin to open a window and render its user interface (5). Upon receiving the X window drawing requests from the X client, the XWeb broker converts them into images to be sent back to the web browser. It is only then that the XWeb broker sends a response (6).

This technique is referred to as a *deferred response*. The response (6) to a request (4) is deferred until the XWeb broker has some information to send to the browser. This technique is necessary to allow asynchronous updates: the X client can update its user interface at any point in time, but because of the client/server relationship of HTTP, the XWeb broker itself cannot forward those updates to

the browser. Those updates are piggy-backed onto the HTTP responses. Note that because the HTTP response is deferred until there is a PDU to be sent to the web browser, this model does not result in busy waiting.

The main event loop of the client-side implementation of the XWeb broker therefore constantly pulls for updates from the XWeb broker (e.g., (4) and (9) in Figure 4). Since the response is deferred until there is something to be sent back, this technique does not revert to busy waiting. In Figure 4, the PDUs 1, 4, 7, and 9 represent HTTP requests, whereas PDUs 3, 6, and 11 represent HTTP responses.

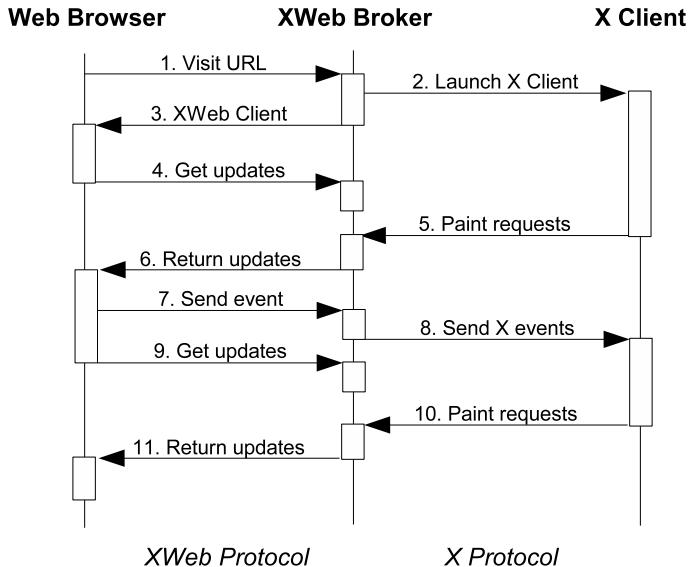


Fig. 4. XWeb Protocol

Whenever the user creates an event (such as pressing a key), the XWeb broker is notified of the event (7). In order reduce the number of mouse events, the web browser can be configured not to send every mouse event. Upon receiving an event, the XWeb broker simply forwards it via the appropriate X-PDU to the X client (8). The X client will react to the event by updating the content of a panel (10). These updates are sent back to the web browser via another deferred response (11).

5 Prototype Implementation

We have done a prototype implementation of XWeb. By implementing the XWeb broker, we had to implement the X Window Protocol as well as the XWeb-specific protocol. In order to leverage Open Source tools as much as possible, we use several freely available Open Source packages.

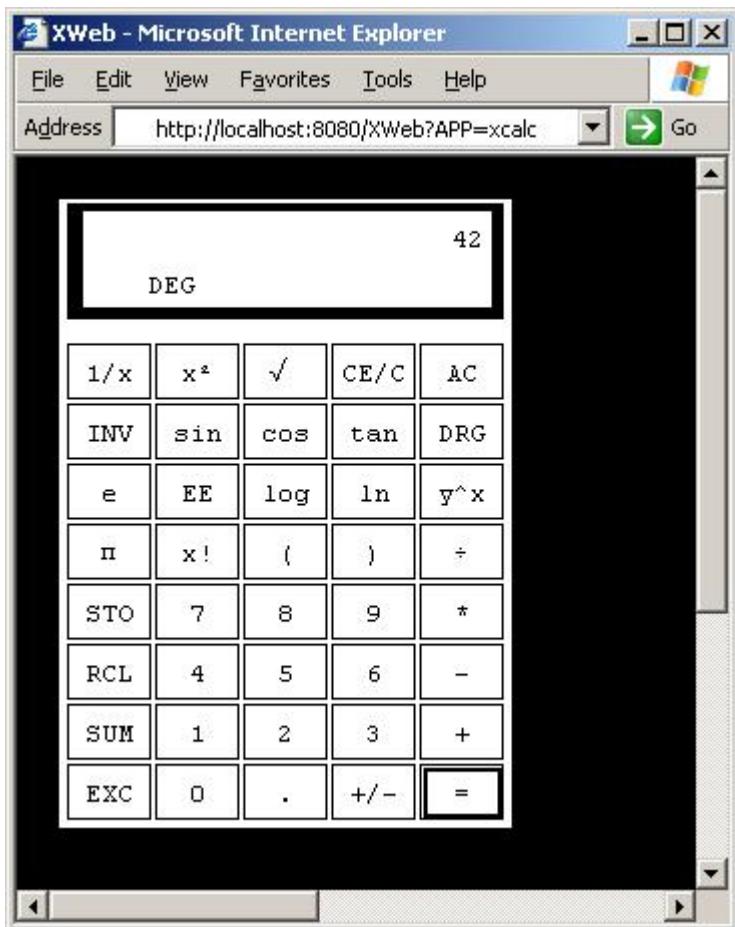


Fig. 5. Running xcalc inside Internet Explorer

As explained in the previous section, the XWeb-broker acts as an X server towards the X clients connected to it. We use the aforementioned WeirdX. Although WeirdX implements an X server in Java, we had the problem that WeirdX is supposed to run on a desktop and it consequently opens a window wherever it runs. In order to suppress this window and to be able to interface with the output, we have implemented our own AWT Toolkit. By implementing the abstract base class `java.awt.Toolkit`, one can intercept the opening of Windows and the creation of widgets. Setting the property `awt.toolkit` during startup of the Java virtual machine, one can override the build-in AWT toolkit. Luckily WeirdX only makes use of very few AWT classes, so that this solution is both simple and elegant. This allowed us to use WeirdX without any modifications.

The communication between XWeb and the web browser is based on HTTP, which means that the XWeb broker acts as a web server towards the browser. We

use the Open Source HTTP engine called Simple (see [2]). As the name implies, Simple is an easy to use, thin web server implementation. The API is modeled after the Servlet specification. Simple is shipped as one JAR file that can easily be linked to other applications such as the XWeb broker in our case.

The client-side of the XWeb protocol is completely implemented in portable JavaScript that runs in all popular browsers. The JavaScript code is loaded into the browser when first visiting the main page served by XWeb through Simple. The user can specify the application to be run via parameters encoded in the URL. E.g., visiting `http://xweb-host.com/XWeb?APP=xcalc` assumes that the XWeb broker is installed on host `xweb-host.com` and it would launch the X application xcalc (an X calculator). Figure 5 shows a screenshot of xcalc running inside Internet Explorer after performing the computation $2*21$. The input focus still shows the mouse on the `=`-key that was clicked last.

We have conducted some performance measurements to estimate the overhead introduced by XWeb. We have used xedit, a popular X editor, for our experiments. The experiments consisted of three phases: starting xedit, typing “Hello World” once xedit has started, and then exiting xedit. Table 3 shows the total number of transferred bytes and the number of PDUs exchanged for each of the three phases. As can be seen, XWeb does incur a significant overhead. This will become apparent when considering how for example a rectangle is drawn.

The X Protocol has a special instruction for drawing rectangles. This instruction contains the geometry of the rectangle and further attributes such as fill color. The corresponding X-PDU is therefore only a few bytes in size. On the XWeb side, the rectangle is actually transferred as a PNG image, which obviously requires more bandwidth. But despite the higher protocol overhead of XWeb compared to the X Protocol, we found that X applications that run inside a browser via XWeb are sufficiently interactive to be usable. Because of the high-bandwidth of most networks one does not notice any significant delays and we were able to operate a complex IDE this way.

Table 3. Comparison of protocol overhead

	Total size (in bytes)		Number of PDUs	
	X	XWeb	X	XWeb
Starting xedit	13.388	90.002	56	168
Typing “Hello World”	7.422	61.935	53	197
Exit xedit	0	16.574	0	35

6 Related Work

We are not aware of any other project that has built a X-to-web protocol bridge. However, there is a different way how our work can be interpreted, which has to do with the way we integrated WeirdX. Recall from the previous section that we leverage the Open Source X server implementation WeirdX by providing

our own AWT Toolkit. The benefit of this approach is that we do not need to make any modifications to WeirdX which certainly has benefits whenever a new version of WeirdX is released. What facilitated our approach is the fact that WeirdX only makes use of two AWT container classes to render its output: `java.awt.Window` and `java.awt.Panel` for which our AWT toolkit provides a replacement implementation.

The consequence is that any Java AWT application that only uses those two classes could be exposed as a web application via XWeb. WeirdX could just be seen as one of those applications. Of course, as soon as an application uses a different AWT class such as `java.awt.Button`, our current XWeb prototype would not work. But it is conceivable to extend the XWeb protocol in such a way that it also supports other widgets such as buttons or listboxes. The web browser as a generic client could provide native support for these widgets. The XWeb protocol would thus support different widget types such as the following:

```
1 <xweb>
2   <create id="ID_2" pid="ID_1" type="button" />
3   <create id="ID_3" pid="ID_1" type="listbox" />
4 </xweb>
```

This path leads to a general application migration framework where any AWT or Swing application could be exposed as a web application (see Figure 6). Several projects – commercial and Open Source – exist that aim at providing an easy migration path for legacy Java applications to web applications. WebCream is a commercial product by a company called CreamTec (see [1]). They have specialized in providing AWT and Swing replacements that render the interface of the Java application inside of a web browser. WebCream makes

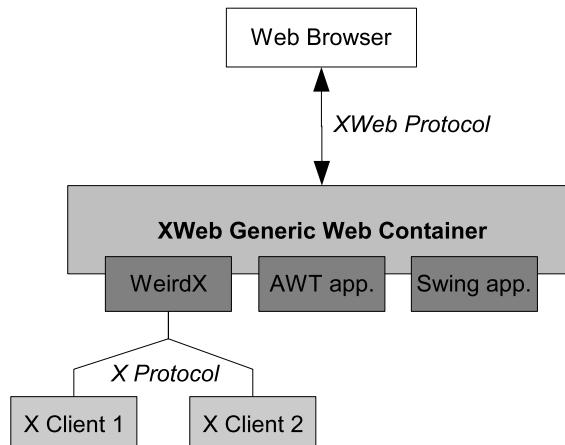


Fig. 6. XWeb Generic Web Container

use of proprietary features of Microsoft's Internet Explorer and therefore only runs inside this browser.

Two Open Source projects, both hosted at SourceForge, follow the same idea of exposing Java desktop applications as web applications. The first one is called WebOnSwing (see [9]). Unlike WebCream, this project is not tailored for a particular browser. One feature offered by WebOnSwing are templates that allow to change the look-and-feel of the application that is rendered inside the browser. Another project with similar features, but not quite as mature, is SwingWeb (see [6]).

Following our earlier argument, it should be possible to run WeirdX as an application in those three projects and achieve the same results as with our XWeb. Our experiments however revealed that this is not possible because none of those projects supports asynchronous updates. In all cases, updates only happen when the user interacts with the user interface, e.g., by pressing a button. An application, such as `xclock`, that produces asynchronous updates, do not update the user interface in any of the three projects mentioned in this section.

7 Conclusions and Outlook

The X Protocol, developed in 1984, has led to a wealth of X applications. E.g., through the Motif library, every Java application is effectively also an X application. While web browsers are ubiquitous nowadays, the same cannot be said of X servers. This paper introduces a migration path that allows access to any X application via any web browser. This can be accomplished without any special browser plugins. Our prototype implementation leverages Open Source tools where possible to implement the protocol bridge efficiently.

XWeb can be considered as a generic web container that could potentially allow one to host any Java Swing or AWT application. With additional work, a comprehensive set of implementations for the other Swing and AWT widgets could be provided. This would make any Java application accessible from any web browser again without the need for special browser plugins. It might also be worth-while to consider different end-user devices such as PDAs. A PDA would thus become the generic client introduced earlier in the paper. The XWeb protocol would eventually become a client/server protocol decoupling different technologies in the same way that the X Protocol decouples the X server from the X client. We have begun this work as outlined in [11].

This framework leads to support for Ajax-applications. Ajax (Asynchronous JavaScript and XML, see [3]) applications run inside a web browser and achieve an interactivity that resembles that of desktop applications. An example of an Ajax application is Google Maps that is entirely implemented in JavaScript. Ajax applications execute part of the application inside the web browser. We are currently investigating to add a code migration framework to the XWeb protocol that is capable of translating Java-byte code instructions to JavaScript (see [10]).

References

1. CreamTec, LLC. *WebCream*. <http://www.creamtec.com/webcream/>.
2. Niall Gallagher. *Simple - A Java HTTP engine*. <http://sourceforge.net/projects/simpleweb/>.
3. Jesse Garrett. *Ajax: A New Approach to Web Applications*. <http://www.adaptivepath.com/publications/essays/archives/000385.php>.
4. Hummingbird Ltd. *Exceed - X Server Support for Microsoft Windows*. <http://www.hummingbird.com/products/nc/exceed/>.
5. JCraft. *WeirdX - Pure Java X Window System Server*. <http://www.jcraft.com/weirdx/>.
6. Tiong Hiang Lee. *SwingWeb*. <http://swingweb.sourceforge.net/swingweb/>.
7. Microsoft Corporation. *Microsoft Java Virtual Machine Support*. <http://www.microsoft.com/mscorp/java/>.
8. The Open Group. *X Window System (X11R6) Protocol*, 1999.
9. Fernando Petrola. *WebOn Swing*. <http://webonswing.sourceforge.net/xoops/>.
10. Arno Puder. An XML-based Cross-Language Framework. In *DOA*, LNCS, Agia Napa, Cyprus, 2005. Springer.
11. Arno Puder. XML11 - An Abstract Windowing Protocol. *PPPJ*, 2005.
12. RedHat. *Cygwin/X - A Free X Server for Microsoft Windows*. <http://www.cygwin.com/xfree/>.
13. WebSideStory. *U.S. Browser Usage Share*. <http://www.websidestory.com/>.

Exploiting Application Workload Characteristics to Accurately Estimate Replica Server Response Time

Corina Ferdean and Mesaac Makpangou

INRIA Rocquencourt, France, BP 105,
78153 Le Chesnay Cedex

Abstract. Our proposition, presented in this paper, consists in the definition of a function estimating the response time and a method for applying it to different application workloads. The function combines the application demands for various resources (such as the CPU, the disk I/O and the network bandwidth) with the resource capabilities and availabilities on the replica servers. The main benefits of our approach include: the *simplicity* and the *transparency*, from the perspective of the clients, who don't have to specify themselves the resource requirements, the *estimation accuracy*, by considering the application real needs and the current degree of resource usage, determined by concurrent applications and the *flexibility*, with respect to the precision with which the resource-concerned parameters are specified.

The experiments we conducted show two positive results. Firstly, our estimator provides a good approximation of the real response time obtained by measurements. Secondly, the ordering of the servers according to our estimation function values, matches with high accuracy the ordering determined by the real response times.

1 Introduction

We place our work in the context of Replica Hosting Systems, which provide access to various Internet services, by guaranteeing the Quality of Service level required by the clients (in terms of responsiveness, availability). Each service is delivered by a group of replica servers running on hosts of various capacities. The main issue in such systems is the selection for each client of the suitable replica server that can best respond to its requests, with respect to the required QoS. We assume that the requests are heterogenous: some requests make a lot of computation; some requests do intensive disk I/O access or generate important traffic on the network; and others have various demands concerning several resources (CPU, disk I/O or network bandwidth). The response time is largely accepted as the ideal metric correlating with the QoS perceived by the clients.

Existing systems estimate response time, based on measurements of the system load, the round trip time, and the available bandwidth. Each of these metrics is good for specific cases. For instance, the round trip time metric works well for static Web documents of small sizes. Also, the available bandwidth becomes the

pertinent metric when the replies returned to the clients contain data of large size. None of the existing solutions could provide estimations that correlate with the real response time, for heterogenous requests combining (arbitrary) the usage of several resources. The main drawback of these solutions is that they ignore the characteristics of the application requests. Within this respect, the estimation metrics lack flexibility, which is needed by the variability of the resource demands for different requests. Another drawback is that they don't exploit the impact of current resource availabilities on the requests response time.

We propose a solution which improves the response time estimation, under the hypothesis of requests heterogeneity. Our approach relies on an estimation function that combines the demands for resources required to serve a request, the intrinsic capacities and the current utilization degree of each candidate replica server host.

We derived this function, by decomposing the response time, resulted from the application execution, into several components: the CPU service time, the disk I/O service time, the network transfer time, the CPU waiting time and the disk I/O waiting time. The service times corresponds to the times during which the CPU, respective the disk I/O resources were used by the request. The network transfer time counts for the time needed to transmit the reply through the network. The waiting times count for the time that the request spends in the ready queue, respective in the disk I/O queue.

The estimation function is the sum of these components, each one being parameterized by the requests workload, and the host and the network resources characteristics (capacity and availability).

The main benefit of the proposed solution is its accuracy. We show experimentally that the estimation function orders correctly replica servers compared to real response time measurements. Furthermore, in several cases, the estimated response time does not diverge too much from the measured response time.

The rest of the paper is structured as follows. Section 2 introduces the system model, together with some base concepts that we consider. Section 3 presents our approach for estimating the CPU waiting time. Section 4 defines the others components of the response time estimation function. Section 5 brings some details on our model within a brief discussion. In Section 6, we validate our approach experimentally, by means of several concrete workloads. Section 7 presents some related work. Finally, Section 8 ends the paper with a conclusion.

2 Background and Base Concepts

2.1 System Model

We associate to a host, noted by $host$, two metrics: the *capacity* and the *utilization*, noted by c , respectively by u . For a particular $host$, we denote its capacity by $host.c$ and its utilization by $host.u$. We consider that each host is characterized by two resources: the CPU and the disk I/O. When comparing the performance of the $host$ with respect to a client, noted by cl , we also consider the network bandwidth resource and the network latency, estimated by the round-trip time. A particular resource capacity is static and characterizes the resource maximal performance. The

resource utilization is dynamic and characterizes the resource contention, when several applications, running concurrently, compete for that resource.

The host capacity is a vector whose components correspond to the CPU, the disk I/O and the network bandwidth resources. We characterize the capacity of a particular resource by the work accomplished by the system within 1 unit of absolute time (e.g. 1 second). In the case of CPU, the capacity is a 2-dimensional vector, counting for the number of floating point operations executed within 1 second and the number of integer operations executed within 1 second. In the case of disk I/O, the capacity counts for the number of MBytes transferred from the disk within 1 second. In the case of the network bandwidth, the capacity with respect to the client cl , counts for the number of Mbits transmitted through the network to cl within one second. For a particular host capacity c , we denote the components presented above by $c.cpu$, $c.io$ and $c_{cl}.net$.

The resource *utilization* metric is expressed in percents (as the measurement unit). For a particular host utilization u , we note the CPU utilization by $u.cpu$, the disk I/O utilization by $u.io$, the bandwidth utilization, with respect to the client cl , by $u_{cl}.net$ and the network latency by $u_{cl}.rtt$.

2.2 Base Concepts

Application Workload. We define the concept of an *application*, as the abstraction for two resource usage scenarios. They correspond to autonomous jobs and to the client's requests, executed by the replica to which the client has been bound.

Each application is assimilated to its *workload*, noted by wk , and which expresses the quantity of resources needed by that application so as to run properly. We define an application workload on three main dimensions: the volume of computation, the quantity of the data accessed from the disk and the quantity of the data transferred through the network. We associate to the former dimension the application metric called the *CPU workload*. It counts for the number of the floating point operations, to be performed by the processor in order to execute the workload. We associate to the second dimension the application metric, called the *disk I/O workload*. It counts for the size of the data read/written from/to the disk, during the workload execution. Finally, we associate to the latter dimension the application metric called the *network workload*. It denotes the size of the data contained within the reply returned to the client. For a particular workload wk , we denote the three components respectively by $wk.cpu$, $wk.io$ and $wk.net$. The measurement unit for the *CPU workload* is 1MFLOP and for the the *disk I/O workload* and for the *network workload* is 1MByte. In the rest of the paper, a triplet of the form $(cpuWk, ioWk, netWk)$ will denote a workload with the CPU component $cpuWk$, the disk I/O component $ioWk$ and the network component $netWk$.

Host Class. We define the abstraction *host class* as a group of hosts which provide similar response times when executing the same application, under 0 utilization (i.e. without contention). A host class may encapsulate the physical characteristics of the processor (e.g. speed, number), of the disk (e.g. storage capacity, I/O bandwidth), the software characteristics concerning the operating system (e.g. the scheduling policy) and the communication protocol stack.

3 Estimating the CPU Waiting Time

We perform a regression-based analysis of the waiting time, where we study the dependence between the waiting times and the current utilization of the resources needed by the workload. Basically, each workload has its waiting time fitting curve, with the baseline points determined by measurements, or by estimations obtained from the measurements of other workloads.

Our experiments showed that the contribution of the waiting time within the global response time becomes significant, especially under the conditions of medium to high resource utilization, when it increases exponentially. Computing the CPU waiting time is very challenging. The main difficulty arises from the fact that the waiting time depends on several parameters (e.g. application workload, resource utilization, system policies). The big issue is to determine the contributions of each parameters to the overall CPU waiting time, and how the parameters interfere with each other. The variability of the parameters values (in time and for different applications) makes this issue even more difficult. In the following, we will define a function, noted by $cpuWt(wk, u)$, that estimates the CPU waiting time for the workload wk , under the host utilization u .

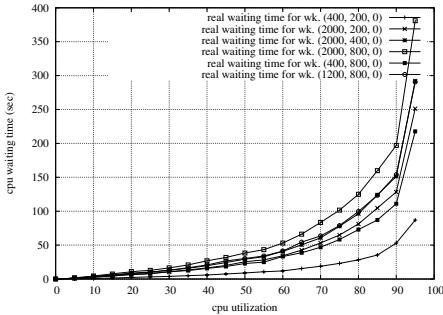
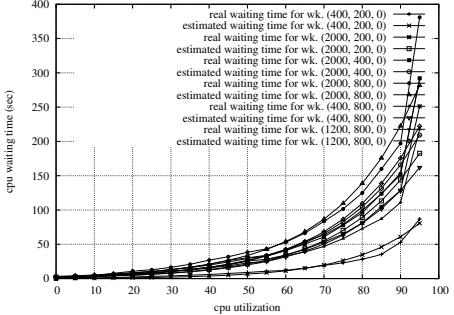
3.1 Empirical Study

In order to gain some insights on the variation of the CPU waiting time, we performed two series of empirical studies, where we measured the real CPU waiting time. In both studies, we fixed the host capacity. In the former study, we fixed the workload and we varied successively the CPU and the disk I/O utilization. We represented graphically the evolution of the CPU waiting time according to the CPU utilization (and given disk I/O utilization values). In the latter study, we fixed the host utilization, and we varied successively the CPU workload and then the disk I/O workload. We represented graphically evolution of the CPU waiting time, according to the CPU workload, respectively to the disk I/O workload.

The experiments to which we refer in this paper have been performed for the host class containing the machines with a Pentium 4 processor with 3GHz, 900MB RAM and running Linux.

Varying Utilization. We began by simply measuring the CPU waiting time for different workloads, while varying the CPU and the disk I/O utilization on the replica host.

Figure 1 shows six graphics for concrete workloads, where we neglected the network workload and we varied the ratio between the CPU and the disk I/O workload. We fixed the disk I/O utilization value to 0. We plotted on x-axis 20 values of the CPU utilization, and on y-axis the corresponding measured values for the CPU waiting time. All the six curves, obtained by unifying the points, follows an exponential evolution, growing more rapidly in the case of superior CPU workloads. This result remains also valid for different values of the disk I/O utilization. Our experiments lead us to the conclusion that, generally, the CPU waiting time grows exponentially, according to both the CPU utilization and the CPU workload.

**Fig. 1.** Measured CPU wt**Fig. 2.** Estimated vs. measured CPU wt

With these elements, the function $cpuWt$ becomes an exponential function with the exponent expressed as a linear function. This function is parameterized by $u.cpu$ and has the form $y = a*u.cpu + b$, where the coefficients a and b depend both on the workload wk and on the utilization value u , under which the waiting time should be estimated for wk . We obtain the following formula:

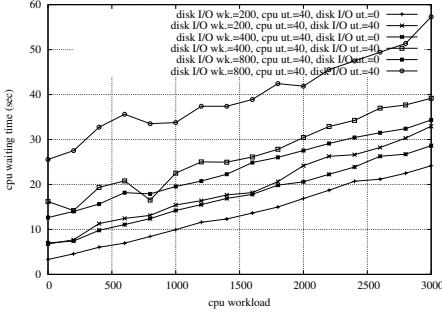
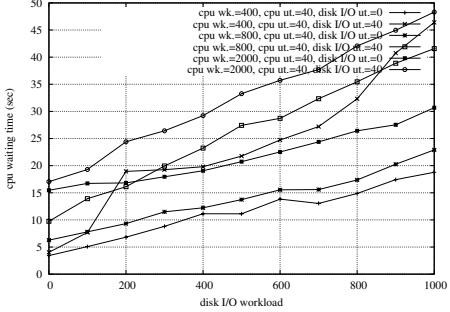
$$cpuWt(wk, u) = \exp(a*u.cpu+b), \text{ where } a = f(wk, u) \text{ and } b = g(wk, u) \quad (1)$$

In order to compute the coefficients a and b , we need the values of the CPU waiting time under several (at least two) utilization values u_i with the same disk I/O component, i.e. $u_i.io = u_j.io$ and $u_i.cpu \neq u_j.cpu$, $i, j = 1, n$, $i \neq j$. We note by d_i the CPU waiting times measured under u_i , $i=1, n$, for wk , i.e. $d_i = measured(wk, u_i)$, $i=1, n$.

At this point, we compute the coefficients a and b by applying the Least Squares Line Fitting method [12]. Basically, this method determines a and b by minimizing the square error, which is $\sum_{i=1}^n (d_i - a*u_i.cpu - b)^2$. We implemented this method within the function $getCoef()$ in Figure 5. This is parameterized by n pairs (x_i, y_i) which serve as the baseline points for the regression-based approximation. In our case, y_i corresponds to the waiting time determined under the CPU utilization value x_i . Precisely, $y_i = \log(d_i)$.

We define in Figure 5 the function $cpuWt_base$, which represents the base building brick of our estimation approach. It contains the core of the regression-based waiting time approximation. It determines the waiting time for the workload wk under the utilization u , by applying the formula (1), where the coefficients a and b are computed using the function $getCoef()$.

In order to validate our approach, we took the same workloads from Figure 1, for which we compared the CPU waiting time computed by the function $cpuWt_base()$ with the real CPU waiting time, obtained by executing the workloads. Figure 2 shows these results, where the estimation curves are very closed to the real measurements curves. These experiments shows very good correlation between the estimated CPU waiting time and the real CPU waiting time.

**Fig. 3.** CPU wt. variation with CPU wk**Fig. 4.** CPU wt. variation with disk I/O wk

Varying CPU Workload or Disk I/O Workload. We performed a second type of measurements, where we studied the evolution of the waiting time when the CPU workload, respectively, the disk I/O workload varies, while the utilization is fixed.

The graphs in Figure 3 show that the CPU waiting time varies linearly with the CPU workload. Precisely it increases linearly with constantly increasing values of the CPU workload. In all the graphs, we fixed the disk I/O workload and the utilization.

The experiments we performed allow us to conclude that, under a given utilization value, the CPU waiting time varies linearly with the CPU workload, while the disk I/O workload is fixed. This result also remains true, when varying the disk I/O workload, with the CPU workload being fixed. Under this scenario, the CPU waiting time also increases linearly, as shown in Figure 4.

With varying CPU respective disk I/O workload, we obtain the following formulas for the CPU waiting time:

$$cpuWt(wk, u) = \alpha' * wk.cpu + \beta' \quad (2)$$

and

$$cpuWt(wk, u) = \alpha'' * wk.io + \beta'' \quad (3)$$

In order to compute α' and β' , we need the CPU waiting times under u , for several (at least two) workloads with the same disk I/O component as wk . We note them by wk_i , such that $wk_i.io = wk.io$, $wk_i \neq wk_j$ and the corresponding waiting times by d_i , $i, j=1, k$, $i \neq j$.

We also apply the Least Squares Line Fitting method. Precisely, we compute α' and β' using the function `getCoef()`, parameterized by $x = \{wk_i.cpu\}$ and $y = \{d_i\}$, $i = 1, k$.

Similarly, in order to compute α'' and β'' , we need the CPU waiting times under u , for several (at least two) workloads with the same CPU component as wk . We note them by wk_i , such that $wk_i.cpu = wk.cpu$, $wk_i \neq wk_j$, and the corresponding waiting times by d_i , $i, j=1, k$, $i \neq j$. We compute α'' and β'' by applying the function `getCoef()`, parameterized by $x = \{wk_i.io\}$ and $y = \{d_i\}$, $i=1, k$.

Conclusion. The formulas (1), (2) and (3) serve as basis for estimating the CPU waiting time, for any workload, under any utilization value. In order to apply these formulas (possibly in various combinations), we need the measurements of the CPU waiting times for some workloads under some utilization values. Basically, we need to define the couples $(wk, n, \{u_i, d_i\})$ which should configure the formula (1), and the couples $(u, k, \{wk_i, d_i\})$ which should configure the formulas (2) and (3) so as to perform the estimation of CPU waiting time for any workload, under any utilization value. The definition of these configuration data represents the object of the next section.

3.2 Configuration Data

Reference Points. We define the *reference points* as a set of host utilizations values, where there are two different utilization values which have the CPU component in common, and other two utilization values which have the same disk I/O component. More formally:

$$P = \{u_i\}, i = 1, n; n \geq 4, \text{ where } \exists i, j, k \text{ and } l, 1 \leq i, j, k, l \leq n, \text{ such that } u_i.cpu = u_j.cpu, u_i.io \neq u_j.io, u_k.io = u_l.io \text{ and } u_k.cpu \neq u_l.cpu.$$

Reference Workloads. We define the *reference workloads* as a set of workloads for which the CPU waiting time has been measured under any reference point. This set is defined so as to contain at least two different CPU workload values and at least two different I/O workload values. Each of these four values is the component of at least two different (reference) workloads, which are included in the set.

More formally:

$$W = \{wk_i\}, i = 1, m, m \geq 4, \text{ where } \exists i, j, k \text{ and } l, 1 \leq i, j, k, l \leq m, \text{ such that } wk_i.cpu = wk_k.cpu, wk_j.cpu = wk_l.cpu, wk_i.cpu \neq wk_j.cpu \text{ and } \exists u, v, w \text{ and } z, 1 \leq u, v, w, z \leq m, \text{ such that } wk_u.io = wk_v.io, wk_w.io = wk_z.io, wk_u.io \neq wk_w.io$$

The rationale behind these formulas will be revealed, while describing the estimation algorithms in the next section.

We associate to each $wk \in W$, the set of the CPU waiting time measures under each reference point from P . More formally:

$M_{wk} = \{(u_i, d_i)\}$, such that $d_i = measured(wk, u_i)$, $i=1, n$. We aggregate all the measurements into the set M , where $M = \{M_{wk}\}$, $\forall wk \in W$. The sets P , W and M are defined for every host class.

3.3 Estimating the CPU Waiting Time for Any Workload, Under Any Utilization

With these elements, we will show how to estimate the CPU waiting time, for any workload wk , under any utilization value u of the *host*. We distinguish between the cases where wk is an element in W or not. In these cases, we name wk a *known workload* respectively a *not-known workload*. In the latter case, if wk has at least the CPU or the disk I/O component in common with a reference workload, we name it a *semi-known workload*, otherwise we name it an *unknown workload*. The

```

getCoef(n, x, y) {
     $sx = \sum_{i=1}^n x_i; sy = \sum_{i=1}^n y_i; ssx = \sum_{i=1}^n x_i^2; sxy = \sum_{i=1}^n x_i * y_i;$ 
     $c = (n*ssx - sx^2); a = (n*sxy - sx*sy)/c; b = (sy*ssx - sx*sxy)/c;$ 
    return <a, b>
}

cpuWt_base(wk, u, n, x, y) {
    <a, b> = getCoef(n, x, y)
    d = exp(a*u + b)
    return d
}

cpuWt_known(wk, u) {
    1. if  $u \in P$ 
        d = measured(wk, u)
    2. else a) get from  $M_{wk}$  n pairs  $(u_i, m_i)$ 
        such that  $u_i \neq u_j$ ,  $u_i.io = u_j.io$  and  $|u_i.io - u.j.io|$  is minimal,  $i, j = 1, n, i \neq j$ .
        b) let  $x = \{u_i.cpu\}$  and  $y = \{m_i\}$ ;  $d = cpuWt_base(wk, u.cpu, n, x, y)$ 
    endif
    return d
}

cpuWt_semiknown(wk, u) { //when wk has the same I/O wk. as two reference wk.
0. get  $wk_i \in W$ ,  $wk.io = wk_i.io$ ,  $i = 1, k$ 
1. if  $u \in P$ ,
     $m_i = measured(wk_i, u)$ ,
    a) let  $x = \{w_i.cpu\}$  and  $y = \{m_i\}$ ; < $\alpha, \beta$ > = getCoef(n, x, y)
    b)  $d = \alpha * wk.cpu + \beta$ 
2. else
    a). get n points  $u_i \in P$ ,  $u_i.io = u_j.io$ ,  $|u_i.io - u.j.io|$  is minimal,  $i = 1, n$ 
    b)  $m_i = cpuWt_semiknown(wk, u_i)$ 
    let  $x = \{u_i.cpu\}$  and  $y = \{m_i\}$ ;  $d = cpuWt_base(wk, u, n, x, y)$ 
endif
return d
}

cpuWt_unknown(wk, u) {
0. get l sets, each one containing k workloads  $wk^i_u \in W$ ,  $wk^i_u.cpu = wk^i_v.cpu$ ,  $i = 1, l$ ;  $u, v = 1, k$ 
1. If  $u \in P$ 
    let  $wk^i.cpu = wk^i_u.cpu$ ,  $wk^i_i = (wk^i.cpu, wk.io)$ ,  $i = 1, l$ 
    a)  $m_i = cpuWt_semiknown(wk^i_i, u)$ 
    b) let  $x = \{wk_i.io\}$  and  $y = \{m_i\}$ ; < $\alpha, \beta$ > = getCoef(n, x, y)
     $d = \alpha * wk.io + \beta$ 
2. else
    a). get n points  $u_i \in P$ ,  $u_i \neq u_j$ ,  $u_i.io = u_j.io$ ,  $|u_i.io - u.j.io|$  is minimal,  $i, j = 1, n$ ,
 $i \neq j$ 
    b)  $m_i = cpuWt_unknown(wk, u_i)$ 
    let  $x = \{u_i.cpu\}$  and  $y = \{m_i\}$ ;  $d = cpuWt_base(wk, u, n, x, y)$ ,  $i, j = 1, n$ 
endif
return d
}

cpuWt(wk, u) {
If  $wk$  is a known workload
     $d = cpuWt_known(wk, u)$ 
else if  $wk$  is a semi-known workload
     $d = cpuWt_semiknown(wk, u)$ 
else  $d = cpuWt_unknown(wk, u)$ 
endif
endif
return d
}

```

Fig. 5. The functions providing the CPU waiting time estimation

following three sections show how the estimation is proceeded if wk is a *known*, *semi-known*, respectively an *unknown workload*. In each case we distinguish between the cases where u is an element of P or not.

The Case of Known Workloads. In the case where wk is a known workload, if u is a reference point from P , we get from M_{wk} the corresponding measure d (obtained under u).

If u is not a reference point, in order to compute the coefficients a and b under the utilization value u , we choose from the set M_{wk} , n pairs (u_i, m_i) , satisfying $u_i.io = u_j.io$, with their common value $u_i.io$ being the closest to $u.io$, among all the reference points couples with the same I/O component. With these elements, we compute $cpuWt(wk, u)$, by applying the function $cpuWt_base()$, parameterized by the baseline points $(\{u_i.cpu\}, \{m_i\})$. We sketched the algorithm for a known workload in Figure 5, within the function $cpuWt_known(wk, u)$.

The Case of Semi-known Workloads. We consider k reference workloads wk_i , which have the same disk I/O workload as wk . 1) If u is a reference point, we get from the sets M_{wk_i} the values m_i representing the CPU waiting times measured under u for wk_i . We compute $cpuWt(wk, u)$ by applying the formula (2).

2) If u isn't a reference point, we get n reference points $u_i \in P$ (with the same disk I/O component) and compute $m_i = cpuWt(wk, u_i)$, using the previous case 1). Finally, we compute $cpuWt(wk, u)$ by applying the regression formula (1) with the baseline points $(\{u_i.cpu\}, \{m_i\})$. We sketched the algorithm for a semi-known workload in Figure 5, within the function $cpuWt_semiknown(wk, u)$.

The Case of Unknown Workloads. We consider l sets, each one containing k reference workloads wk^i_u , such that: $wk^i_u.cpu = wk^i_v.cpu$, $i=1, l$, $u, v=1, k$. Let $wk^i.cpu = wk^i_u.cpu$, $wk^i = (wk^i_u.cpu, wk.io)$

The estimation algorithm is presented within the function $cpuWt_unknown(wk, u)$ in Figure 5. 1) If u is a reference point, the algorithm proceeds in two steps. In the first step, we estimate the CPU waiting time m_i for the semi-known workload wk^i , by applying the previous primitive $cpuWt_semiknown()$.

In the second step it estimates the CPU waiting time d for the workload wk , by applying the regression formula (3), with the baseline points $(\{wk^i.cpu\}, \{m_i\})$.

1) If u is not a reference point, we get n reference points $u_i \in P$, with the same disk I/O component, i.e. $u_i \neq u_j$, $u_i.io = u_j.io$. We compute $m_i = cpuWt(wk, u_i)$, $i=1, n$ using the previous case 1). Finally, we estimate the CPU waiting time d for wk , by applying the function $cpuWt_base()$, parameterized by the baseline points $x = \{u_i.cpu\}$ and $y = \{m_i\}\}$.

4 The Other Response Time Components

4.1 The Disk I/O Waiting Time

We followed a similar strategy for estimating the disk I/O waiting time, as in the case of the CPU waiting time. Because of the paper space limit, we only sketch

it here, by transposing the formulas (1), (2) and (3) to the case of the disk I/O waiting time. We note the corresponding function by $ioWt(wk, u)$, estimating the disk I/O waiting time for the workload wk under u . We use the same configuration data, consisting in the reference points P , the reference workloads W and the associated measurements M .

We performed the same two series of measurements, Firstly, we measured the disk I/O waiting time, while varying the value of the disk I/O utilization. The curves obtained follow this time a linear evolution, that we adopted when deriving the formula transposing (1) to disk I/O waiting time:

$$ioWt(wk, u) = a*u.io + b.$$

We consider n points $u_i \in P$ such that $u_i.cpu = u_j.cpu$ and $u_i.io \neq u_j.io$ and d_i is the measure of the disk I/O waiting time under u_i , $i=1, n$. The coefficients a and b are computed using the function $getCoef()$, parameterized by $x = \{u_i.io\}$ and $y = \{d_i\}$.

The second type of measurements showed that the disk I/O waiting time varies linearly with the workload. We obtain the following formulas similar to (2) and (3): $ioWt(wk, u) = \alpha'*wk.io + \beta'$ and $ioWt(wk, u) = \alpha''*wk.cpu + \beta''$.

We consider k reference workloads $wk_i \in W$ with the same CPU component, and d_i is the value of the disk I/O waiting time measured for wk_i under u_i , $i=1, k$. The coefficients α' and β' are computed by applying the function $getCoef()$, parameterized by $x = \{wk_i.io\}$ and $y = \{d_i\}$.

The coefficients α'' and β'' are computed similarly to the coefficients α' and β' , using reference workloads with the same disk I/O component.

As in the case of the CPU waiting time, we developed the base function $ioWt_base(wk, u)$ and we distinguished similarly between a *known*, *semi-known* and *unknown* workload, which we treated within the functions $ioWt_known(wk, u)$, $ioWt_semiknown(wk, u)$, respective $ioWt_unknown(wk, u)$.

4.2 The Other Components of the Response Time

The *CPU service time* is estimated by dividing the CPU workload by the CPU capacity, expressed as the number of floating point operations per second. The *disk I/O service time* is estimated by dividing the disk I/O workload by the disk I/O capacity.

The *network transfer time* is also estimated by means of a regression based technique. We distinguish between bandwidth-intensive vs. non-bandwidth intensive network workloads. The former workloads have the transfer time dominated by the bandwidth, while the latter workloads have the transfer time dominated by the round-trip-time. We discuss here only the transfer time for the bandwidth-intensive workloads. A workload is bandwidth intensive if its value is greater than the maximal bandwidth between a host and a client. In this case, the transfer time is given by the formula $nt(wk, host, cl) = wk.net/(a*bw + b)$, where bw is the available bandwidth between $host$ and cl . We compute a and b , for some known workloads, for which there are available some measurements of transfer time, under some bandwidth utilization values. These measurements are used as the baseline points within the Least Squares Line Fitting method, which gives a and b . For the

$$\begin{aligned} \text{cpuSt}(wk, c) &= wk.cpu/c.cpu \\ \text{ioSt}(wk, c) &= wk.io/c.io \\ \text{rt}(wk, host, cl) &= \text{cpuSt}(wk, host.c) + \text{ioSt}(wk, host.c) + \text{cpuWt}(wk, host.u) + \text{ioWt}(wk, host.u) + \text{nt}(wk, host, cl) \end{aligned}$$

Fig. 6. Formalizing the estimation function

other unknown workloads, the transfer time is computed using the transfer time of known workloads under the same utilization value. This computation relies on the observation that the transfer time increases linearly with the network workload, under a given bandwidth utilization value.

With these elements, the response time function becomes the sum between the *CPU service time*, the *disk I/O service time*, the *CPU waiting time*, the *disk I/O waiting time* and the *network transfer time*. Figure 6 translates this definition into three mathematical formulas.

5 Discussion

The application CPU workload can also be expressed in number of integer operations. In this case, we convert it to a CPU workload expressed in number of floating point operations, using the CPU capacity of a given (reference) host. Precisely, we divide the CPU workload by the host capacity in terms of number of integer operations per second, and multiply the result by the host capacity in terms of floating-point operations per second.

An important question is how to choose the reference points. We conducted some experiments in order to answer to this question. We varied the utilization reference points, both in terms of number and distribution over the range values [0, 100]. For each reference points set, we computed the coefficients a and b , for seven different workloads. For each workload, we computed the estimation error over a testing set with 20 points. We showed the results in Table 1. The main conclusion is that increasing the number of reference points doesn't improve the accuracy, unless the points are well chosen, i.e. uniformly distributed around "the middle".

Our approach has two main possible drawbacks. The former is the overhead of monitoring the resource utilization, while the latter points the difficulty of specifying quantitatively the workload. With respect to the former drawback, we propose heuristics for estimating the resources utilization, in a given time-interval, based on the active applications allocated to the concerned host and the host's capacity. With respect to the latter drawback, we propose a solution based on *workload classes*, which aggregate workloads with similar demands for a particular resource. We define classes for each of the three resources quantitatively -by means of interval values- or qualitatively -by means of semantic keywords (e.g. multimedia).

Another issue is related to the variability of the utilization metric, due to concurrent applications which interference with the benchmarked workload. We have obtained recently some preliminary results, promising that the estimator can tolerate a certain degree of staleness in the utilization values used, without degrading too much the accuracy of estimation.

Table 1. Estimation errors when varying the reference utilization points

Reference utiliz. points	Error for wk (400, 200, 0)	Error for wk (2000, 200, 0)	Error for wk (2000, 400, 0)	Error for wk (2000, 600, 0)	Error for wk (2000, 800, 0)	Error for wk (400, 800)	Error for wk (1200, 800)
10 20	241798.20	9078.83	2882.08	7205.66	12688.70	4694.38	10323.13
20 30	23.51	100.77	195.83	130.55	167.97	363.20	128.94
80 90	19.49	90.46	102.91	123.30	134.59	81.01	99.71
20 60	40.49	116.10	120.43	156.56	174.76	76.57	136.80
10 20 30	3482.33	941.28	854.83	1307.19	1306.20	1323.72	1033.57
30 50 70	36.15	101.70	133.55	144.66	148.14	103.53	118.01
20 40 60	38.13	112.93	114.00	149.02	165.76	73.64	131.06
10 40 80	17.50	71.97	85.61	93.93	105.83	61.38	75.25
10 15 20 25 30	2014.77	824.78	818.11	1097.96	1103.53	977.14	876.76
20 40 60 70 80	36.73	105.52	118.32	138.10	157.01	85.16	117.79
20 20 40 60 80	36.41	104.94	112.33	135.12	155.92	80.15	114.91
10 20 40 60 70	22.55	73.54	90.97	100.95	106.92	61.21	83.10
10 20 40 60 80	18.02	79.16	91.71	104.79	117.22	63.44	85.11

6 Experimental Validation

In order to validate our estimation approach, we performed experiments for various concrete workloads. We used a *simulator*, capable to generate application workloads. Precisely, it instantiates a test application with the same workload as the real application. This workload is generated by means of basic operations like: float multiplications, read/write disk accesses, send/receive socket accesses. We used the reference points set $P = \{u_i\}$, $u_i.cpu, u_i.io \in \{0, 20, 40, 80\}$. We performed the baseline measurements (of CPU waiting time, respective of the disk I/O waiting time) under 3 CPU respective disk I/O utilization values $\{20, 40, 80\}$ (this vector is the argument x of the function *getCoef()* in Figure 5).

For a given workload *wk*, the experimentation follows the following scenario. We vary the values of the CPU, disk I/O and network bandwidth utilization, for a given *host*. Under a given utilization value, i.e. the triple (CPU utilization, disk I/O utilization, network bandwidth utilization), we determine the couple containing the response time estimated with our approach and the real response time measured by executing the workload. For n utilization values (i.e. n experiments), we obtain the set D with n samples. Each sample i is defined as the couple (*estimated response time*, *real response time*), noted by ert_i , respectively rrt_i . More formally: $D = \{(ert_i, rrt_i)\}, i=1, n$ and $ert_i < ert_{i+1}, \forall i i=1, n-1$. We draw the graph associated to D , plotting the values ert_i on x-axis and the corresponding values rrt_i on y-axis.

We define the *value estimation error* for each sample in D . We note it by $valEr_i$, where $i=1, n$, $valEr_i = \min(100*|ert_i - rrt_i|/rrt_i, 100)$.

We define the *variation estimation error* for each two successive samples in D . We note it by $varEr_i$, where $i=2, n$, $varEr_i = \min(100*(rrt_i - rrt_{i-1})/rrt_{i-1}, 100)$, if $rrt_i > rrt_{i-1}$, and 0 otherwise.

6.1 Experimenting System Workloads

We begin the validation of our response time estimation approach, by considering only system workloads, where the network component is 0. Specifically, we

experimented our approach for three workloads $(2000, 800, 0)$, $(2000, 400, 0)$ and $(400, 800, 0)$, varying the ratio between the computation demands vs. the disk I/O demands. For each workload, we considered both the cases where it was *known*, -in which case the estimation relies directly on its reference points-, respectively *unknown*, -in which case the estimation relies on four reference workloads. The reference workloads used were: $(400, 200, 0)$, $(400, 2000, 0)$, $(3000, 200, 0)$ and $(3000, 2000, 0)$.

The Figure 7 shows the results in the case of the workload $(2000, 800, 0)$, Figure 8 for the workload $(2000, 400, 0)$ and Figure 9 for the workload $(400, 800, 0)$. We represented 44 points on each graph. Each point on a graph, corresponds to a particular CPU and disk I/O utilization value, and is represented by plotting on the x-axis the estimated response time and on y-axis the real measured response time (both of them obtained under that utilization value).

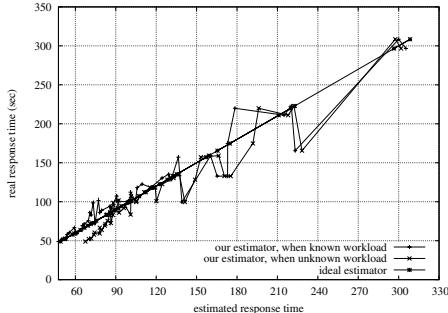
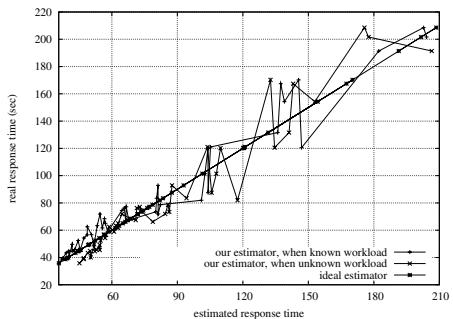
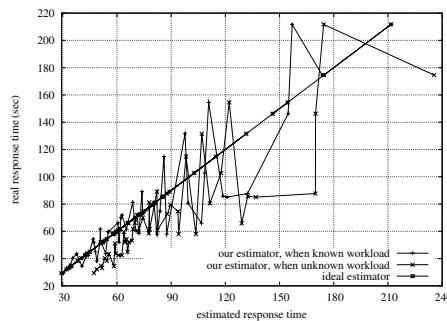
In each case, we compared the response time estimated by our approach with respect to the ideal estimation (represented by the real response time). One can see that in each graph, the curves corresponding to the estimated response time vs. the real response time are close to each other (being closer in the case of the known workloads). Another important observation is that the variation of the real response time matches with good accuracy the variation of the estimated response time (i.e. the estimated vs. real time correlation curve is mostly monotone increasing).

6.2 Experimenting Complete Workloads

In this section we show the results of using our approach for complete workloads. The experimentation follows a scenario similar to the case of system workloads, except that in this case we also vary the network bandwidth utilization, when determining the graph points. We represented 64 points on each graph.

The graphs in Figures 10 and 11 show the response time estimation for the workloads $(2000, 800, 20)$, respective $(2000, 800, 80)$, considered known vs. unknown. One can see that the monotony of the real response time matches, in most cases, that of the estimated response time (i.e. in most cases, a bigger estimated response time corresponds to a bigger real response time). Comparing the results of our estimator with an ideal estimator, one can see that there is an over-estimation of the response time, but this still remains under reasonable limits. We also have noticed that the bad estimation happens for big values of the utilization. When the values of utilization are small or even reasonably high, the estimation works fine.

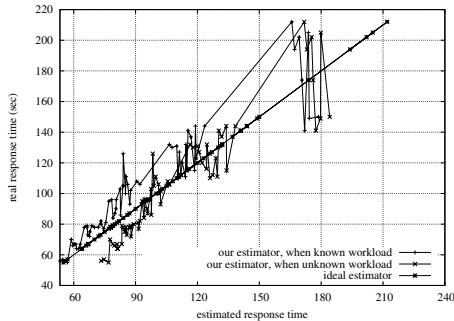
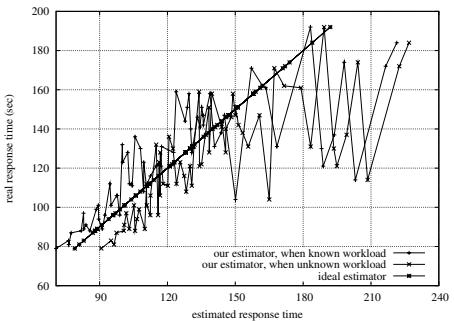
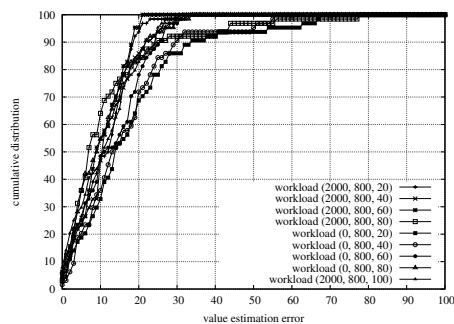
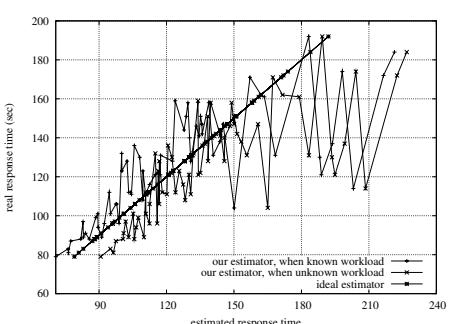
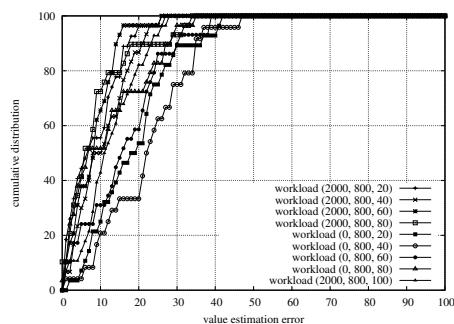
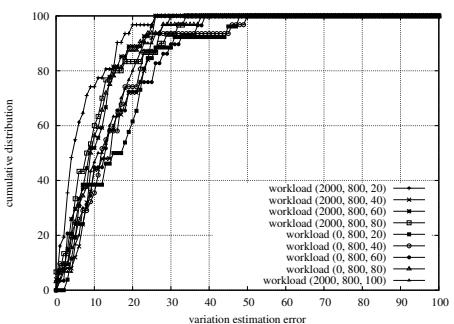
We adopted two means to validate our model: by performing a regression based analysis of the estimation results and by studying the cumulative distribution of the estimation errors. In the former evaluation scenario, we studied the linear relationship between the estimated response time and the real response time, using as the baseline a set of 44 points resulted from measurements. Precisely, we determined the coefficients of the line $y = a \cdot x + b$, where x represents the estimated response time and y is the real response time. We represented the results in Table 2. Each row corresponds to a workload, considered known vs. unknown. If the estimation were perfect, $a = 1$ and $b = 0$. One can see that the results are good

**Fig. 7.** Estimation for wk. (2000, 800, 0)**Fig. 8.** Estimation for wk. (2000, 400, 0)**Fig. 9.** Estimation for wk. (400, 800, 0)**Table 2.** Coefficients a and b for various workloads, known and unknown

Workload	coef. a and b, when wk. known	coef. a and b, when wk. unknown
(2000, 800, 0)	0.91 12.08	0.99 -8.45
(2000, 400, 0)	1.01 3.94	1.15 -13.77
(400, 800, 0)	0.94 2.46	0.87 -4.71
(2000, 800, 20)	1.00 10.74	1.12 -17.40
(2000, 800, 80)	0.56 55.44	0.61 39.90

enough, except for the last case, when the network workload is equally important as the system workload. Except this case, the estimation works fine.

With respect to the second evaluation scenario, Figures 12 and 13 shows the cumulative distribution for the value estimation errors, considering the case of known workloads, respectively unknown workloads. A point on the graph is defined by plotting on x-axis the value estimation error, and on y-axis the percentage of points whose error is inferior to the x-axis value. These results show that the estimation accuracy is satisfactory. Figures 14 and 15 shows the cumulative distribution for the variation estimation errors. A point on the graph is defined by

**Fig. 10.** Estimation for wk. (2000, 800, 20)**Fig. 11.** Estimation for wk. (2000, 800, 80)**Fig. 12.** Value estimation error for known workloads**Fig. 13.** Value estimation error for unknown workloads**Fig. 14.** Variation estimation error for unknown workloads**Fig. 15.** Variation estimation error for unknown workloads

plotting on x-axis the variation estimation error, and on y-axis the percentage of points whose error is inferior to the x-axis value. These results show a good correlation between the estimated response time and the real response time.

7 Related Work

7.1 Survey of Existing Estimation Metrics

We investigated existing work on server selection and load balancing topics, so as to identify the mechanisms used to estimate the expected response time. These mechanisms rely basically on metrics approximating response time, with different degrees of accuracy. We classify these metrics into static and dynamic. The static metrics include basically the geographical distance, as used in [10], the number of router hops, as used in [2, 8, 9, 14, 16], and the number of AS hops, as used in Globule [11]. We further classify the dynamic metrics into basic and composite metrics. The former are obtained by direct measurements, while the latter are computed by formulas which make use of basic metrics. The basic metrics include the network latency (as given by round trip time, for example), as used in [2, 6, 9, 14, 16], the available bandwidth, as used in [4], in Web Server Director [15], in [5, 7, 8], the HTTP request latency [14, 16], the transfer time of a given (probe) file [17] or server load [3, 7, 17]. Examples of composite metrics include *PredictedTT*, used in [6], *S-percentile*, used in [16], *Weighted Total Response Time* used in [5], and the metrics used in Radar [13], in the anycast service [17] and in [1]. The *S-percentile* combines the average and the variance of previously observed latencies. The *PredictedTT* metric sums the round-trip time and the transmission time, which is the document size divided by the available bandwidth). The *Weighted Total Response Time* [5] has a formula equal to *Predicted Transfer Time* multiplied by a weighting factor aimed to prioritize local traffic. The Radar's metric combines proximity in terms of number of hops and the replica load in terms of number of active connections. The anycast service's metric multiplies two factors, counting for the server load, respectively for the network path characteristics.

We conclude the investigation of existing works, by pointing that they limit the environment conditions considered (e.g. in terms of utilization) and they also ignore the request characteristics, in terms of resource demands. Our approach exploits these parameters more deeply, as we showed that they have a significant impact on the response time.

8 Conclusion

The main results of our work consist in a simple and reliable response time estimation function and a method for applying it to different application workloads. The function combines the application demands for system and network resources (CPU, disk I/O and network bandwidth) with the capacities and availabilities of these resources on the concerned hosts. The main benefit of our approach is the estimation accuracy, by exploiting quantitatively the resources needed by the application vs. the resources provided by the replica servers. Current work continues in two directions, which consists in exploiting the heuristics for estimating the resource utilization (instead of monitoring) and defining the workload classes

(instead of a precise workload specification). We also intend to study how the estimation is able to deal with the machines heterogeneity (e.g. so as to compare the response times estimated on Intel vs. Apple machines).

References

1. O. Ardaiz, F. Freitag, L. Navarro, Improving the Service Time of Clients using Server Redirection, 2001.
2. Cisco Distributed Director, Cisco Content Routing Protocols, white paper, 2000.
3. V. Cardellini, M. Colajanni, P. S. Yu, Geographic Load Balancing for Scalable Distributed Web Systems, In Proc of 8th MASCOTS, 2000.
4. R. Carter, Performance Measurement and Prediction in Packet-Switched Networks: Techniques and Applications, Ph.D. thesis 1997.
5. M. Chen, W. Mao, Anycast By DNS Over Pure IPv6 Network, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 2001
6. M. Crovella and R. Carter, Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks, In Proceedings of IEEE INFOCOM, 1997.
7. J. Dilley, B. Maggs, Globally Distributed Content Delivery, IEEE Internet Computing, vol. 6, no. 5, 2002.
8. Z. Fei, M. Ammar, E. Zegura, Multicast Server Selection: Problems, Complexity and Solutions, IEEE Journal on Selected Areas in Communication, Special Issue on Internet Proxy Servers, vol. 20, no. 7, pp. 1399-1413, 2002.
9. J. Guyton and M. Schwartz, Locating nearby copies of replicated internet servers. In Proceeding of ACM SIGCOMM'95, 1995.
10. J. Gwertzman and M. Seltzer. The case for geographical push cashing. In Proceeding of the 5th Workshop on Hot ZTopic in Operating Systems, 1995.
11. G. Pierre, Maarten van Steen, A.S. Tannenbaum, Dynamically selecting optimal distribution strategies for Web documents. IEEE Transactions on Computers 6(51), 637-651, 2002.
12. Mathworld, <http://mathworld.wolfram.com/LeastSquaresFitting.html>.
13. M. Rabinovich, A. Aggarwal, Radar: A Scalable Architecture for a Global Web Hosting Service, WWW8/Computer Networks, 31(11-16):1545-1561, 1999.
14. M. Sayal, Yuri Breitbart, Peter Scheuermann, Radek Vingralek, Selection algorithms for replicated web servers. In Proceeding of the Workshop on Internet Server Performance, 1998.
15. Radware, Web Server Director, white paper, 2002.
16. R. Vingralek et al., Web++: A System For Fast and Reliable Web Service, Proceedings of the USENIX Annual Technical Conference, Sydney, Australia, pp. 171-184 (June 1999).
17. E. Zegura, M. Ammar, Z. Fei, and S. Bhattacharjee, Application-layer anycasting: a server selection architecture and use in a replicated web service, IEEE/ACM Transactions on Networking, vol. 8, no. 4, pp. 455-466, Aug. 2000.

Automatic Introduction of Mobility for Standard-Based Frameworks

Grégory Haïk², Jean-Pierre Briot¹, and Christian Queinnec¹

¹ Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie,
4, place Jussieu , 75252 Paris Cedex 5 France

{Jean-Pierre.Briot, Christian.Queinnec}@lip6.fr

² Thalès Communications / SC2, 1–5, avenue Carnot,

91883 Massy Cedex France

Gregory.Haik@fr.thalesgroup.com

Abstract. The computerization of industrial design processes raises software engineering problems that are addressed by distributed component frameworks. But these frameworks are constrained by a set of antagonistic constraints, between performances and reusability of the components. In order to take up this challenge, we study how mobile code technology enables the improvement of performances without harming the components' reusability. Our approach relies on a transparent, totally automatic introduction of mobility into the programs. This transformation is a local optimization which is based on a static analysis. It is implemented within a compiler. An experimental study shows how the approach can be helpful for increasing the efficiency of the framework, enabling the usage of standards that – as for today – lack of efficiency.

1 Introduction

System engineering, industrial design and manufacturing have been totally transformed, since 1950, by the raise of performances of computers and software, especially Computer-Aided Design (CAD) systems and numerical simulation programs. But users are still waiting for appropriate integration frameworks [12, 5] that would link these programs all together. Still today, engineers have a lot of manual and repetitive, work in order to make their software environment adapted to the particular needs of a specific project.

But industrial design integration frameworks are facing difficult problems : on the one hand, they need to enable a good level of reusability – hence they need to rely on standardized contracts and interfaces –, and on the other hand, the applications built upon these frameworks need to be efficient. Unfortunately, using standards tends to produce unefficient resulting applications, especially in a distributed environment. In order to bridge the gap between performance issues and conformance to standardized interfaces, we have proposed some compilation techniques for automatic introduction of mobility into programs interacting with software components. Mobile code is used as a mean to improve locality, and

consequently performances [2]. By raising up the performances of such programs, software architects can rely on standards that are usually considered as unusable because of the poor performances they imply.

In this paper, we present the results of our research for efficient execution of distributed, standard-based integration frameworks, applied to industrial design applications [9]. The next section presents the motivations (section 2), and leads to the description of our approach (section 3). We will then compare our approach to related works (section 4). The following section will describe more deeply the analysis and compilation techniques and present our prototype. We will finally depict the experimental study that shows the tangible benefits of our techniques (section 6).

2 Motivations

This section will first present an example of integration framework for industrial design. Based on this example, we will explain why such frameworks tend to be distributed, and why they should rely on standards. We will conclude by showing that the usage of standards raises efficiency problems.

2.1 An Example of Framework for Industrial Design: SALOME

SALOME [16] is a ministry-led consortium (RNTL) aimed at defining a component-based framework for integration of software systems involved in industrial design, such as CAD systems, meshing software, numerical solvers, databases of physical properties, visualisation and post-treatment software. Figure 1 shows a snapshot of the user interface of SALOME. In this example, the user has imported the geometry of a ship and meshed it with the help of a meshing component. The user would then typically assign materials to the ship geometry such as, for instance, carbon composite and aluminium; apply forces to the structure for folding and/or torsion; load a solver of structural mechanics – embedded in a SALOME component; and then check by computation whether the ship structure is complying to its requirements. The user could also load a fluid mechanics solver in order to check the ship's structure in conditions of navigation. As we can see, since the usage scenarios of SALOME cannot be predicted, the framework includes a program interpreter (actually a Python interpreter), so that the user can customise the components integration according to his/her specific usage.

2.2 The Need for Distribution

One can wonder why such component-based frameworks are distributed. Indeed, some companies have developed integration frameworks for industrial design in a monolithic, mono-process configuration. However, there are several reasons to make such frameworks distributed. The first reason is the typical size of data and computation timings needed by these applications. If a single-station implementation is feasible for small problems, it becomes unrealistic when the user wants

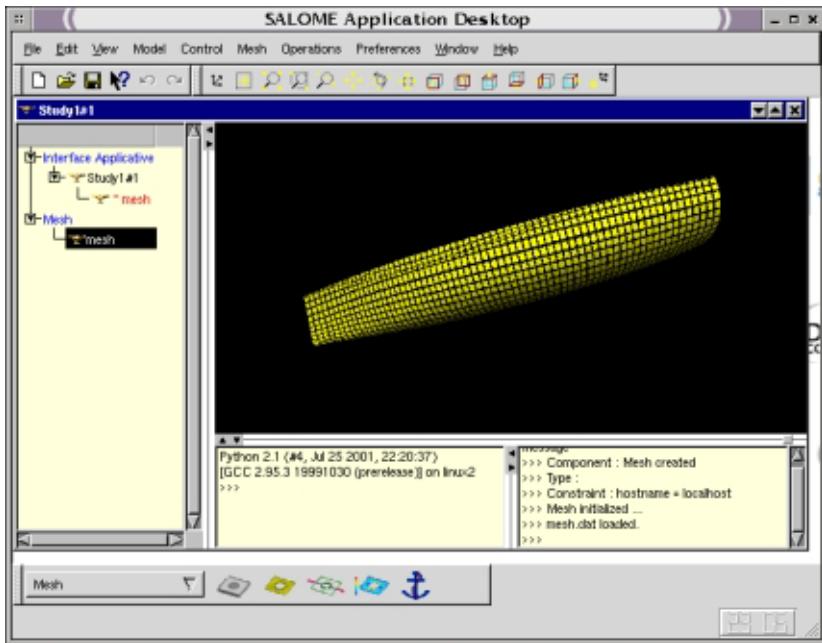


Fig. 1. Meshing of a ship structure in SALOME

to refine the simulation, which leads to bigger problems. Then, distribution can help to manage more data (distributed storage), and to speed up computation (parallelism). The second reason is that the software systems encapsulated in components may have constraints on the underlying hardware and/or operating system. For instance, the visualization tool would require a good 3D graphics processing unit, a numerical solver could be specifically developed for a particular data parallel architecture, and so on. Then distribution is a mean to easily satisfy a set of constraints expressed by the different components. Finally, such component-based industrial design frameworks are aimed, at least in mid-term, to enable a large scale co-operation between engineers in different, distant locations.

2.3 The Need for Standardization

We will develop here a short remark about the definition of contracts and interfaces between the components, and between a component and the framework. On the one hand, considering the number of different components to be integrated, the number of human actors (users, vendors, integrators...), the number of different usage scenarios, it seems necessary to standardize the interfaces upon which the frameworks are built. Indeed, without this effort, the components are condemned to be incompatible one with the other. On the other hand, all the actors of this application domain are not necessarily enthusias-

tic regarding standards. There is a blatant lack of confidence in major standardization organizations like OMG. Some people believe that the standards produced by such organizations are unusable : if they are too exhaustive, then no software component can implement them completely; if they are too small then they become useless. Although this description is caricaturized, the question raised by the inherent flaws of the standardization processes must be addressed thoroughly. To illustrate this lack of confidence, let us examine the case of CAD Services.

2.4 An Example of Standardization Process: *OMG CAD Services*

In 2001, the OMG has established a working group aimed at creating a set of standard interfaces for accessing CAD systems encapsulated in a software component. The list of contributors proves that this standardization effort meets a real users' need (Boeing, Nasa, GE, Ford, ...). CAD systems vendors, such as 3DS and OpenCascade, have also participated in this effort. The group has worked for two years, and frequent meetings were held all around the world, where harsh discussions and debates took place : there exists a natural antagonism between users and vendors, since users push the standard to be as exhaustive as possible while vendors want it thinner so they can implement it more easily. Moreover, there is another natural antagonism between the different vendors, who try to make the standard ontology as close as possible to their own products ontology.

As a result of these technical and logistical impediments, the final result of this effort is disappointing. The working group has only achieved to re-

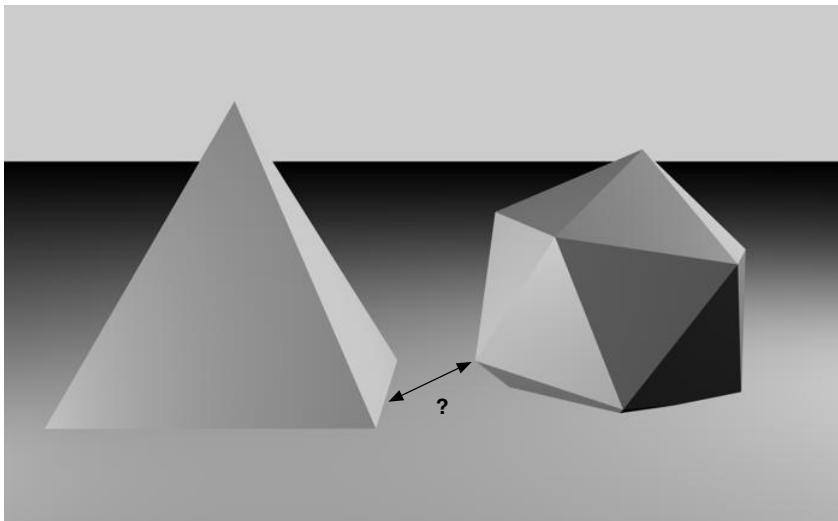


Fig. 2. A Pair of solids hosted by a CAD server

lease a standard – called *CAD Services* [14, 3] – for geometric shapes warehouses. Shapes are defined in terms of very basic and fine-grained concepts such as points, vertices, faces, and solids. Moreover, CAD Services hardly includes any of the most common algorithms usually applied to these data. Consequently, when users want to manipulate geometrical data hosted by such a CAD server, they need to implement their algorithms on the client side, while fine-grained data is accessed on the server side. In a distributed environment, this is particularly inefficient. For instance, consider a CAD server hosting the two solids shown in Figure 2. Consider that the solids have an electrical potential difference, and that the user wants to know whether an electric arc can occur between the two solids. For this, he has to compute the minimal distance from one solid to the other. But unfortunately, the standard does not include any operation for this algorithm. As we will see in section 6, we have implemented this example : a client is located in Paris, while the CAD server is in Nice (1000 km, ping round trip time 20 ms). With a simplified algorithm implemented in the client and a pair of very small shapes made of 117 vertices hosted in the server, the computation almost takes 40 minutes.

3 Our Approach

Our solution to this problem is to transform in a completely automated manner the client program so that it sends a piece of mobile code, that we call a *mobilet*, to the server. This transformation is made in an optimizing compiler. The input of the compiler is exactly the same source code than for the example described

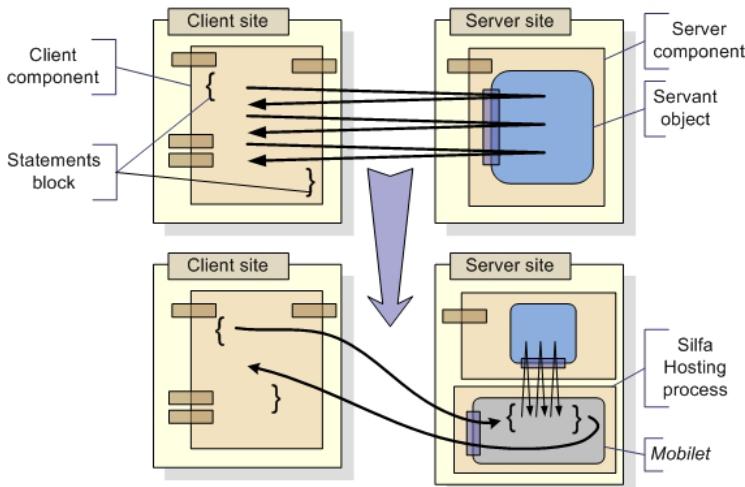


Fig. 3. Transformation of client code by the introduction of a mobilet

above. The computation of the physical distance decreases from 40 minutes to just over 2 minutes. The optimized code is about 18 times faster than without our compiling techniques.

The main challenge addressed by our optimizing compiler is to identify the interesting pieces of code to be embedded in a *mobilet* and executed remotely. The result of the transformation is illustrated on Figure 3. The original code performs a large number of remote interactions between the client-side and the server-side components. By executing a *mobilet* containing part of the client code – in a dedicated hosting process located on the server side – the remote interactions are transformed into local interactions, resulting in a sensible improvement of performances.

4 Related Works

Our approach is a variant of *automatic partitioning*. In a similar way as *automatic parallelization* transparently analyzes and transforms sequential programs in order to discover opportunities for introducing parallelism [13, 1, 6], *automatic partitioning* (or *automatic distribution*) tends to analyze and transform centralized programs in order to discover opportunities for introducing distribution. During the past few years, some research has been conducted in this domain, as reviewed below.

JavaParty [8] is an extension of Java that automatically transforms regular Java classes into remotely accessible ones. It also provides migration of these classes' instances. Users specify which objects are to be made remote/mobile by tagging their classes with a new modifier (keyword `remote`). When an object is migrated, it accesses Java API on the host where it is executed : inputs and outputs are taken from and sent to the destination host, which can be interpreted as a problem of correctness of the transformation.

Doorastha [4] is quite similar to JavaParty, but differs from it on the following issues : Java syntax is not modified, and users insert pragmas in Java comments. Thus, compatibility with genuine Java compilers is preserved. Moreover, in Doorastha's object migration model, calls to `System.out` are forwarded to the original JVM's console, which denotes consideration for the problem of correctness. Still, in Doorastha, there is no tracking of *every* such location-dependent primitives of the Java API, which leads to inconsistencies.

Pangaea [17] is a distribution system for Java applications that works with both JavaParty and Doorastha as back-ends. It is based on a static analysis of Java programs that computes an approximation of the runtime object graph. The Pangaea user specifies, through a graphical user interface, which objects are tied to which hosts. From this specification, the system computes a good placement of every other objects among the anticipated runtime population, by minimizing the number of repeated remote calls.

J-Orchestra [19] and **Addistant** [18] are two automatic partitioning systems for Java bytecode. J-Orchestra distributes Java classes among the network (with the help of the user, like in Pangaea), using statistics gathered by a runtime profiler – in a calibration stage – in order to make placement decisions. The profiler is applied to the non-transformed program in order to evaluate the computational flows between classes. Moreover, classes that contain platform-specific code in native format are considered anchored to their host : they can not be made mobile. A semi-automatic process ensures that no such class will eventually be run on the wrong machine. Regarding to our notion of correctness, J-Orchestra is the only partitioning system that provides a sound mechanism for distributing code.

Coign [11] is a partitioning system for applications made of COM components. It combines typical usage scenarios, application and network profilers in order to make placement decisions, by scrutinizing inter-component communications. As Coign is designed for client-server distribution, it constrains GUI calls to remain on client side, while data storage calls remain on the server side.

Compared to the related works reviewed above, one should notice the three main contributions of our approach. First, our grain of mobility introduction is atomic : our compiler can make mobile every single statement of the original program, while systems reviewed above can only distribute COM components or Java objects. Our fine-grain mobility introduction enables to take advantage of automatic distribution for slices of code for which previous techniques would have been constrained by the including component/object.

Second, we provide a formal framework, partially based on first-class environments semantics [15], that asserts the conditions of a sound automatic distribution. Other approaches focussed on real-world languages such as Java sources, Java bytecode or binaries. Thus, the validity of the program transformations reviewed could not easily be formally proven¹, and we even consider that the majority of them are unsound. We do not present our formal framework in this paper. Interested readers should refer to [9, 10].

Finally, there is an important difference in the goal of related research and ours : previous works have focussed on the distribution of a *stand-alone*, centralized program that is to be executed on a network of computers. Distribution is seen as a motivation in itself, coming from the suboptimal usage of computer resources of laboratories and companies or from the fact that a particular application should be divided between a client side and a server side. On the contrary, we do not consider stand-alone programs to be candidates for transformation : we study programs that interact with other computers by RPC-like techniques. Here, distribution is not seen as a goal in itself, but rather as a mean to minimize the physical distance between a set of distributed resources and their client code.

¹ Because of the technicalities involved in managing real-world languages in a formal manner.

5 A Compiler for Automatic Introduction of Mobility

This section describes the basic techniques of automatic introduction of mobility into communicating sequential programs. We will first present on a simple example the static analysis, which is aimed at (*i*) identifying the pieces of code the compiler should transform and (*ii*) gather the information required by the transformation itself. We will then describe our compiler prototype, and discuss how these techniques should be extended to enable the compilation of higher-order languages.

5.1 Identification of Relevant Pieces of Code

The left part of Figure 4 shows a simple program computing the sum of each column of a remote matrix *m*. It is made of two nested loops : the external one (variable *i*) ranging over lines, the internal one (variable *j*) over columns.

When the optimizing compiler is given such a program, it first identifies the non-movable primitives. In this example, there is only one : `printInt`. It is not movable because its effect depends on the host it is executed on : we can not move it without modifying the semantics of the program. For a given language, there are many non-movable primitives, and we suppose we statically know all of them. The next operation performed by the compiler is to propagate the non-movability property : every piece of code from which a non-movable primitive is accessible, is

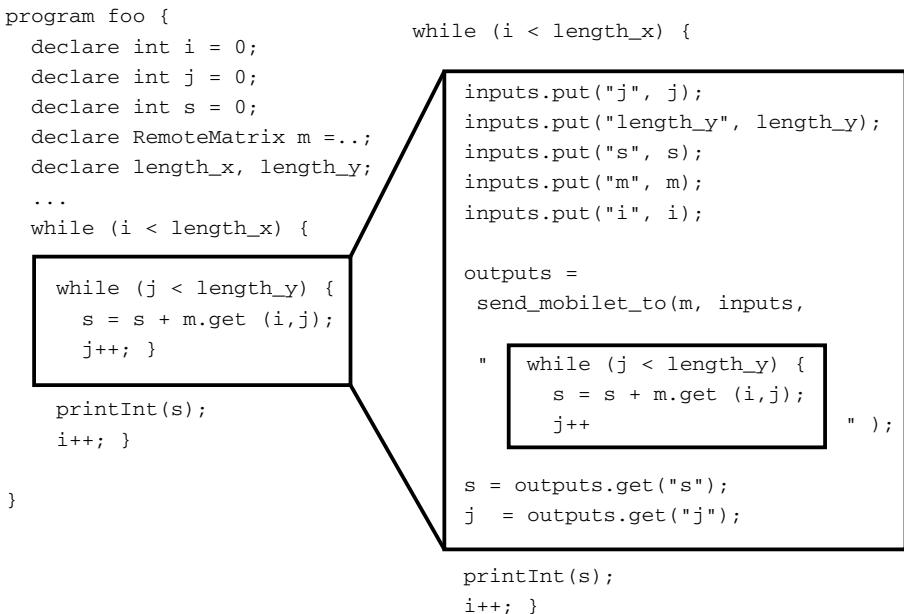


Fig. 4. Principles of code analysis and transformation

also marked as non-movable. Thus, the remaining, not marked, pieces of code can be moved without compromising the global behavior of the program.

Then, the compiler empirically identifies a good candidate for remote execution. A good candidate is a loop, not marked as non-movable, that performs method calls to one or more remote objects. In the example of Figure 4, the external loop does not fit this definition, since it is not movable, but the internal loop – boxed on the figure – does. The compiler will transform it so it will be embedded in a *mobilet* in order to be remotely executed on the server hosting the matrix m .

For this, the compiler needs to compute the set of input and output variables of the *mobilet* : the inputs are the variables read by the boxed statement², and the outputs are the written variables. In our example, the inputs of the mobilet are variables j , `length_y`, s , m , and i . Outputs are variables s , and j .

The compiler has now collected all the relevant data to produce the transformed code, as shown on the right side of Figure 4. The boxed statement on the left side is transformed into the outer-most box on the right side. The compiler generates the filling of a variable-value pair list (namely `inputs`) containing the input chunk of the environment. It then introduces a remote execution primitive (`send_mobilet_to`) that will create a *mobilet* containing the original code – boxed on the left side –, with the input environment chunk (`outputs`) that will be restored after the remote execution. The *mobilet* will be sent to the computer hosting the remote object reference m involved in the loop.

5.2 Managing Multiple Remote Calls and References in a Mobilet

When there are more than one remote object reference in the code to be sent, the following question arises : can the compiler decide what is the appropriate host to receive and execute the *mobilet* ?

Our approach, implemented in the prototype, is to perform a *regularity test* at runtime on the references collected in the piece of code : before sending the *mobilet*, the runtime support inspects the address of a maximum of 10 references. If, and only if, all the addresses are the same, the decision is taken to send the *mobilet* to that host. This technique has a quite small, bounded overhead. Its flaw is that the compiler does not necessarily know all the references at the time when the decision is taken. For instance, in an expression like `o.m1().m2()`, there is no possibility to anticipate, before the computation of the expression, which machine hosts the receiver of $m2$. There are other cases like this one.

Another approach is to intercept every remote call on one (or more) execution of the piece of code (without introduction of mobility), and to perform a statistical breakdown of the addresses of the receiving hosts. An empirical model can then decide, on the basis of this breakdown, whether it sounds valuable to send the *mobilet* for the next executions and where to send it. In comparison to the previous approach, the advantage of this one is that all the remote calls are taken into account. Its disadvantages are that it may be more costly because of the interceptions, and that it needs a calibration stage.

² Omitting the non-free variables, *i.e.* those declared in the sub-block of the statement.

5.3 A Compiler and Analyser Prototype

Silfa : A Dedicated Toy Language. We have implemented a prototype of a compiler for automatic introduction of mobility. It compiles a dedicated toy language called Silfa – a simple imperative sequential language. Silfa users can define procedures and functions, manipulate data arrays and invoke remote operations on CORBA objects. We have decided to study a toy language since the size of the analysis code is proportional to the number of grammar rules that generate the programs : Silfa grammar is made of 45 rules, while Java's has more than 200 rules. Notice that Silfa is not object-oriented, that it is not concurrent, and that there are no pointers, and particularly no pointers to function. We will examine in section 7.1 how to extend the compiler for a higher-order language.

Compiler Implementation. The Silfa analyser and compiler is illustrated on Figure 5. When given a program, the compiler generates a set of Java source code programs, one for the main program, and several mobilets for remote execution. A regular Java compiler generates afterwards Java executable bytecode. The Silfa compiler user interface has an option to disable the generation and connection of the *mobilets*, so we can benchmark the benefits of introduction of mobility.

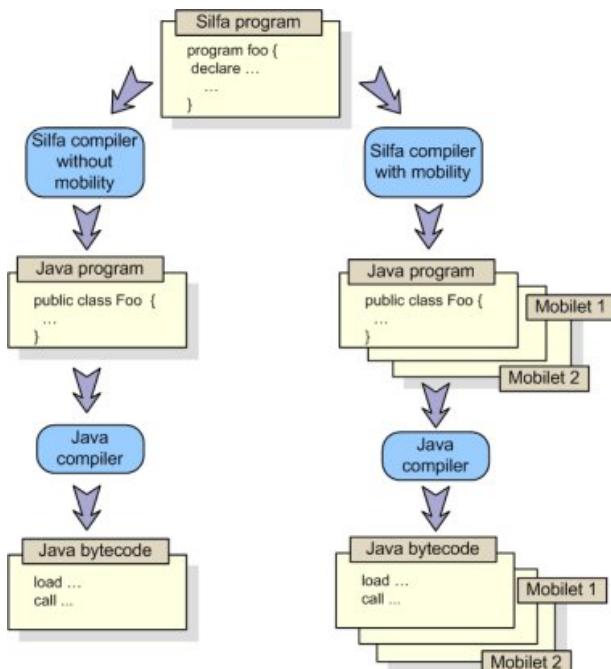


Fig. 5. Prototype of a Mobility Introduction Compiler

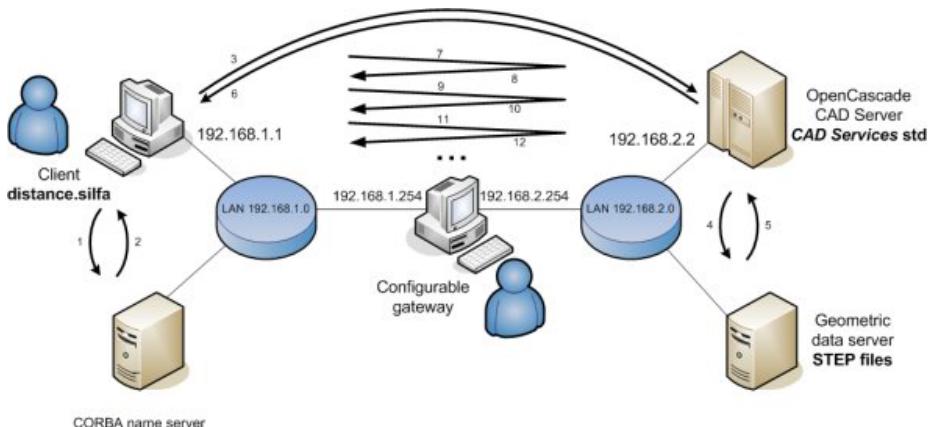


Fig. 6. Experimental network for variation of the latency

6 Experimental Study

The experimental study aims at showing that automatic introduction of mobility can bridge the gap between efficient and reusable distributed architectures.

6.1 Tested Applications and Experimental Settings

The first experiment we have conducted addresses the following question : there is a number of iterations performed by the *mobilets* beyond which the cost of remote execution is prohibitive, and leads to worst performances; isn't this number too big ?

We have designed a small example to answer this question : the example is made of a server program hosting a remotely accessible two-dimensional matrix of integers, and a client program writing and reading the entire matrix. We noticed that for a matrix of 22×23 , with a round-trip latency³ of 1 ms , the transformed program is 50% faster than without introduction of mobility. Since such a matrix is very small compared to the typical data of industrial design applications, and since this latency is also quite small⁴, we conclude that the iteration number threshold above which introduction of mobility is profitable is small enough for the targeted applications.

The second and most relevant experimentation tends to show how introduction of mobility can speed up applications that are sensitive to latency. For this, we have built an experimental network with a configurable latency : it is made of two different LANs linked by a customized Linux gateway implementing a packet delaying mechanism. This enables variation of the RTT from $225\text{ }\mu\text{s}$ (a regular LAN) to 20 ms (a high performance WAN from Paris to Nice).

³ Round-trip time measured by *ping*.

⁴ An RTT of 1 ms is a limit timing between the performances of a bad LAN and a good WAN.

Figure 6 shows this experimental network in the settings of a 3D distance computation through the interfaces of CAD Services, as presented in section 2.4.

We have actually used wireframe shapes only, in order to simplify the 3D distance algorithm. The sequence executed by client program `distance.silfa` is the following :

- Perform a request to the naming service to get a reference to the CAD server (steps 1 and 2);
- Ask the CAD server to load the two wireframe shapes (117 vertices each) from a file repository (steps 3–6);
- Range over the vertices of the first shape and compute its distance with each of the vertices of the other shape (steps 7–);
- Print the minimum of all computed distances.

This program is very simple and the size of the data (two shapes of 117 vertices) is very small. A real application would certainly generate much more remote interactions. Thus, if the experimental results are good for this example, we can expect that they would be even better for a real application.

6.2 Experimental Results

Figure 7 shows the execution timings of the 3D distance computation with and without introduction of mobility, for a latency varying from 1 to 20 ms. The first

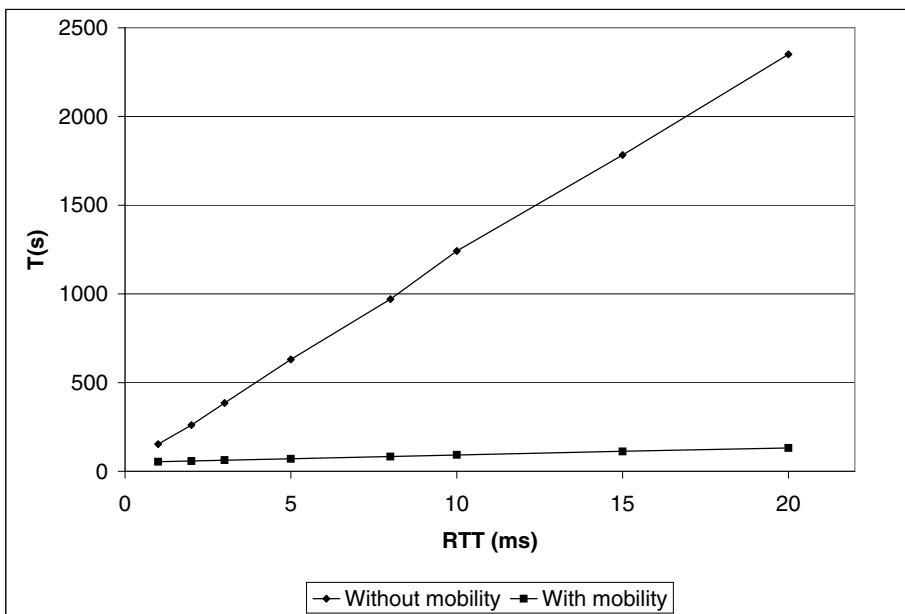


Fig. 7. Execution time *vs.* round-trip time latency (1 – 20 ms)

remark is that latency has a very important impact on global execution timings of the program compiled without introduction of mobility : it varies from 153 s for an RTT of 1 ms , to 2350 s for 20 ms . With our compiling techniques, the impact of latency, if not completely null, is dramatically decreased. As a result, introduction of mobility is very profitable, especially for big values of latency – and still 20 ms is not such a big latency : from Paris to Tokyo, we have noticed an average ping of more than 300 ms , between two well connected universities.

Although our compiler is designed for big latencies, it also produces a sensible optimization, for this application, in a LAN setting : with a ping of 225 μs , the optimized program is 25% faster than without introduction of mobility. Conversely we noticed, as expected, that the possible speed-up depends on the applications : for the example of the remote matrix presented earlier, a ping of 225 μs leads to higher performances – about 20% faster – *without* introduction of mobility. It is also the case for two other applications we have tested, a ranging over a list and a ranging over a three-dimensional array.

7 Future Works

We consider two directions for continuing our research on introduction of mobility. The first direction addresses the compilation of higher-order languages. The second direction addresses the the decision process for sending a *mobilet*.

7.1 Towards a Compiler for Higher-Order Language

When the analyser computes, for instance, the movability property of expressions and statements throughout a program, it needs to consider the user-defined subroutines that are called by these expressions and statements. Indeed, a statement is not movable if it calls a non-movable procedure or function. Thus, the very first step of the analysis is to associate, for each expression and statement, the set of called subroutines. This is a context-sensitive call graph. For Silfa programs, this call graph is easy to compute. But when the language is provided with pointers to functions, first-class functions, or objects, the call graph construction is more difficult. For instance, consider an object-oriented language : for a given method call $o.m()$, the code that will be actually executed depends on the dynamic type of o . The knowledge of the static type of o and the name m of the method is not enough to determine the associated code. There are techniques to statically build such object-oriented call graphs [7] : the analyser alternates data flow analysis (to anticipate the dynamic types of the object references) and control flow analysis (to refine the call graph), until reaching a fixed point. We have not implemented such techniques in our prototype : we have considered that our interprocedural analysis is sufficient for our proof of concept of introduction of mobility.

Other issues have to be addressed in order to compile a mainstream object-oriented language like Java, especially concurrency and synchronisation management, and exception support.

7.2 Improvement of the Dynamic Decision for Remote Execution

We have already presented in section 5.2 a sharper decision process for choosing the appropriate machine that would receive and execute a particular *mobilet*. In addition, the experimental study shows that the optimizing compiler, for very small latencies, may produce a slower code. We can decrease this risk by deciding whether to send the *mobilet* on the basis of the value of a metrics, that would take into account some of the different parameters affecting the speed-up : latency – of course –, by also the number of remote interactions transformed into local ones, the available computation power of the client and the server, and any other parameter that a deeper experimental study would show as relevant.

8 Concluding Remarks

We have defined a static analysis method for introducing, in a totally automated manner, mobility primitives in imperative, sequential, communicating programs. This method is implemented in an optimizing compiler designed for a simplified language. Experimental results show that the optimized programs are dramatically more efficient than non-optimized programs, as soon as the latency gets over a small threshold. Moreover, the correctness of the program transformation is formally proven [8, 9].

We believe that these kind of compilation techniques can benefit to standardization process. Indeed, we have shown that a CAD Services client can be about 18 times faster when interacting with a server located at 1000 km. One could argue that if the CAD Services standard would have included a 3D distance operation, this example would have been meaningless. It is true but it is not the point. The standard itself is not to blame : we have shown that the designers cannot anticipate all the usage scenarios of the standard. And algorithms that are not anticipated must be implemented in the client.

If automatic introduction of mobility reaches a mature status, standard designers could partly rely on the compiler : standards could be more concise, focussing on data exchange, and would let the compiler move the non-anticipated algorithms from the client side to the server side for fast execution. It would renew confidence in standardization organizations like the OMG, since the components based on their standards would be both reusable and performant.

References

1. J.R. Allen and K. Kennedy. *PFC : A program to convert Fortran to parallel form*. Technical Report MASC TR82-6, Department of Math. Sciences, Rice University, Houston, 1982.
2. David M. Chess, Colin G. Harrison, and Aaron Kershenbaum. Mobile Agents: Are they a good idea? In *Mobile Object Systems – Toward a Programmable Internet, LNCS 1222*, pages 25–47, Berlin, Germany, 1997. Springer-Verlag.
3. R. Claus and M. Kazakov. CAD Services: An industry standard interface for mechanical CAD interoperability. In *Proceedings of Concurrent Engineering 2003 conference*, ISBN 90-5809-622-X, 2003.

4. M. Dahm. The doorastha system. *Technical report B-1-2000, Freie Universität Berlin*, 2000.
5. Mohamed E. Fayad, Douglas C. Schmidt, and Ralph E. Johnson. Application Frameworks. In Mohamed E. Fayad, Douglas C. Schmidt, and Ralph E. Johnson, editors, *Building Application Frameworks*, pages 29–54, New York, 1999. Wiley Computer Publishing.
6. Paul Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1):23–53, 1991.
7. David Grove, Greg DeFouw, Jeffrey Dean, and Craig Chambers. Call graph construction in object-oriented languages. In *ACM Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA)*, pages 108–124, 1997.
8. Grégory Haïk. Introduction of Mobility for Distributed Numerical Simulation Frameworks : a Formal Study. Technical Report LIP6 2003/008, Université Pierre et Marie Curie – Laboratoire d’Informatique de Paris 6, Paris, France, 2003.
9. Grégory Haïk. *Introduction de mobilité dans les applications de conception industrielle*. PhD thesis, Université Pierre et Marie Curie (Paris 6) – Laboratoire d’Informatique de Paris 6, 2005.
10. B. Haumacher, T. Moschny, and M. Philippsen. <http://www.ipd.uka.de/javaparty>.
11. Galen C. Hunt and Michael L. Scott. The coign automatic distributed partitioning system. In *Operating Systems Design and Implementation*, pages 187–200, 1999.
12. Ralph E. Johnson. Components, frameworks, patterns. In *ACM SIGSOFT Symposium on Software Reusability*, pages 10–17, 1997.
13. D.J. Kuck, Y. Muraoka, and S.C. Chen. On the number of operations simultaneously executable in fortran-like programs and their resulting speed-up. In *IEEE Transactions on Computers C-21*, pages 1293–1310, 1972.
14. Object Management Group. Computer Aided Design Services Specification, OMG document formal/05-01-07, version 1.2, 2005.
15. Christian Queinnec and David De Roure. Sharing code through first-class environments. In *Proceedings of ICFP’96 — ACM SIGPLAN International Conference on Functional Programming*, pages 251–261, Philadelphia (Pennsylvania, USA), 1996.
16. Salome. <http://www.salome-platform.org/>.
17. André Spiegel. Automatic distribution in pangaea. *Proceedings of Workshop on Communications-Based Systems, CBS 2000*, April 2000.
18. Michiaki Tatubori, Toshiyuki Sasaki, Shigeru Chiba, and Kozo Itano. A bytecode translator for distributed execution of “legacy” Java software. *Lecture Notes in Computer Science*, 2072:236–256, 2001.
19. E. Tilevich and Y. Smaragdakis. J-orchestra: Automatic java application partitioning. In *Proc. of European Conference on Object-Oriented Programming (ECOOP)*, Malaga, June 2002.

Empirical Evaluation of Dynamic Local Adaptation for Distributed Mobile Applications

Pablo Rossi and Caspar Ryan

School of Computer Science & IT,
RMIT University Melbourne, Victoria, Australia
`{pablo, caspar}@cs.rmit.edu.au`

Abstract. Distributed mobile applications operate on devices with diverse capabilities, in heterogeneous environments, where parameters such as processor, memory and network utilisation, are constantly changing. In order to maintain efficiency in terms of performance and resource utilisation, such applications should be able to adapt to their environment. Therefore, this paper proposes and empirically evaluates a local adaptation strategy for mobile applications, with ‘local’ referring to a strategy that operates independently on each node in the distributed application. The strategy is based upon a series of formal adaptation models and a suite of mobile application metrics introduced by the authors in a recent paper. The experiments demonstrate the potential practical application of the local adaptation strategy using a number of distinct scenarios involving runtime changes in processor, memory and network utilisation. In order to maintain application efficiency in response to these changing operating conditions, the system reacts by rearranging the object topology of the application by dynamically moving objects between nodes.

1 Rationale

It has been recognised that applications with distributed components differ from their traditional non-distributed counterparts along a number of important dimensions including communication type, latency, concurrency, partial versus total failure, and referencing/parameter-passing strategies [1].

Rapid advances in hardware technology have given current laptop machines the processing power of servers only a few years old, with smaller mobile devices such as Intel XScale based PDA’s now having CPU’s running at hundreds of megahertz with 64MB or more of program memory. Phone technology is also rapidly advancing with current generation phones able to run application code using standardised platforms such as Java 2 Micro Edition (J2ME) [2], Symbian OS [3], and Microsoft .NET Compact Framework [4]. As such, an increasingly diverse range of potential devices, including desktop PCs, laptops, PDAs and smartphones, are capable of running object oriented application code in a virtual machine environment, and thus actively participate as part of a distributed mobile application.

With such a diversity of possible devices, previous challenges such as communication latency, failure mode and concurrency are now complemented by the issue of maintaining efficiency and quality of service in a significantly more heterogeneous networking and execution space.

Although these challenges appear daunting, Ryan and Perry [5] demonstrated in an empirical study that there are substantial benefits to be realised by end-users and application service providers, through better utilisation of the computing power of client side devices. The primary caveat to such an approach is minimising the additional developer effort required to produce applications with fatter or adaptive smart clients that can take advantage of increasing client-side computing resources.

In order to address the challenge of writing distributed applications for heterogeneous environments whilst minimising the amount of additional developer effort, a context aware adaptive mobile application framework called MobJeX [6] is being developed, in conjunction with the work described in this paper, as part of the Applications Program of the Australian Telecommunications Cooperative Research Centre (ATcrc). MobJeX is a software development platform with a middleware component enabling dynamic application adaptation based on object mobility in response to environmental changes detected by runtime resource monitoring.

In this context, application adaptation refers to the ability of an application, or the underlying middleware, to modify its behaviour in response to changes in environmental context, (e.g. available network bandwidth or CPU load). In the case of MobJeX, adaptation is achieved via object mobility, in which individual system components, potentially down to the discrete object level, can migrate through the system whilst maintaining location transparency via remote object references.

Given the challenge of writing applications that can adapt to run more efficiently in diverse operating environments, and given the existence of a prototypical application framework that includes fully functional system monitoring and transparent object mobility, this paper is concerned with deriving, implementing and testing a local adaptation algorithm using a number of realistic scenarios applied to an actual distributed application running within the MobJeX framework. The concept of local adaptation refers to a strategy whereby the algorithm operates independently at each system node in order to coordinate the adaptation of the distributed application.

The rest of this paper is organised as follows: Section 2 briefly summarises a suite of metrics and models for mobile applications developed by the authors in recent work. This serves as a formal basis for the derivation of the local adaptation algorithm presented in this paper. Section 3 provides a literature review of existing work in application adaptation of distributed systems, particularly those featuring object mobility as a means of adaptation. Section 4 details the derivation and operation of the new adaptation algorithm itself, while section 5 describes a series of empirical studies using a real application running on the MobJeX framework in order to demonstrate the potential practical application of the algorithm using a number of common scenarios. Section 6 finishes with a summary, conclusions and a discussion of opportunities for future work.

2 Background

In previous work [7], the present authors proposed, and mathematically modelled using metrics, a number of software and efficiency attributes likely to have an impact on the execution of mobile applications. This suite of metrics and their representative models were identified from a critical analysis of the problem domain after a review

of the metrics literature, and empirically validated through the formulation of concrete hypotheses expressing the intuitive relationships between the software and efficiency attributes. Table 6 in the Appendix lists this set of attributes and associated metrics, and provides a summary of the relationships among them.

The authors also identified a practical application of the metric based models, involving runtime application adaptation; particularly in terms of runtime object topology, in which the clustering of application objects and placement of object clusters to nodes varies in response to changing environmental conditions. Two decision-making strategies were proposed. Firstly, a global adaptation strategy whereby the application is analysed as a whole, with optimisation performed in terms of mapping object clusters to nodes. Secondly, a more dynamic and reactive technique called local adaptation involves individual nodes moving one object at a time to other hosts when either a performance or resource utilisation threshold is met.

In this previous study a preliminary adaptation algorithm was presented in order to illustrate the practical application of the metrics and models within an existing mobile application framework called MobJeX [6]. This paper extends that work by deriving and implementing a new local adaptation algorithm that provides a substantial amount of flexibility in terms of preference and weighting given to individual quality related attributes. This algorithm, is described in detail in section 4, and evaluated empirically in section 5 using a number of live scenarios involving different adaptation policies applied to a real mobile application running on the MobJeX framework.

Before describing the new algorithm it is first appropriate to review existing work on application adaptation, particularly that involving object mobility. Therefore, the following section examines a number of previous studies, identifying aspects that have been incorporated into the work described in this paper, as well as highlighting limitations and thus providing further rationale for the algorithm presented in section 4.

3 Literature Review – Application Adaptation

Adaptation for distributed applications has received significant research attention in the last few years. Earlier approaches focused on adaptation for client-server applications, [8-11], while more recent efforts address adaptation for parallel applications [12] or generic distributed applications where individual components reside on peers [13-20].

The common feature of these papers is the monitoring of the execution environment to detect resource utilisation and capacity changes and trigger an adaptation mechanism to improve performance. The most commonly monitored resource attributes, which are measured either in terms of total capacity, utilisation or both, are (in order of frequency): network, memory, processor and battery. In addition, some of the studies consider the type of information exchange between [10, 11], or the nature of interaction among, distributed application components [12, 13, 18].

Note however that none of the existing approaches consider the impact of adaptation on resource utilisation and performance at the same time, as is done by the adaptation algorithm described in section 0 and evaluated in section 0 of this paper. Furthermore, none of these proposals take into account software attributes of application components nor do they make decisions based on a formal model using empirically validated metrics, again as is the case of the strategy presented herein.

The actual mechanisms employed to achieve adaptation include object migration [12, 13], the selection of alternative methods [16, 19], the substitution of object implementations [8, 17, 21], and middleware reconfiguration [9, 14, 15, 20]. In some cases the information exchanged by components is adapted, rather than the components themselves [10, 11]. How this is done can depend on factors such as whether the information is binary or text-based. Another adaptation strategy involves using location information to place servers close to the clients [18].

The approaches that employ object migration as the adaptation mechanism [12, 13], are the most relevant to the adaptation algorithm presented in this paper. However, in contrast to this new strategy, neither of the existing approaches is transparent from the perspective of the application developer, the significance of which is described in the following paragraph. In fact with the exception of [10, 14], which are not concerned with adaptation via object migration, none of the existing adaptation approaches are application transparent.

Jing [22] identifies three broad classes of application adaptation. First is *laissez-faire adaptation*, a strategy in which the application is entirely responsible for triggering and implementing adaptation. Second is *application aware adaptation*, wherein applications explicitly interact with middleware services to facilitate adaptation. Last is *application transparent adaptation*, which is the most desirable but most difficult to achieve in practice. In this case, both the decision to trigger adaptation and the strategy for executing it are controlled independently of the application via middleware. This type of adaptation, which is the subject of this paper, is the most appealing from the perspective of the software developer, since the software can be implemented using conventional techniques while still realising the potential benefits of adaptation.

Another point of distinction between the reviewed studies is that there is no clear tendency with regards to the scope of adaptation. Some approaches adapt the application as a whole [9-11, 14-16], while others include the ability to adapt discrete application components [8, 12, 13, 17-21]. The latter approach is more flexible since it offers more adaptation options and finer granularity, but is more complex since it requires a more sophisticated implementation mechanism.

Finally, none of the previous approaches, with one exception [12], distinguish between local (or decentralised) and global (or centralised) adaptation as discussed in the previous section. Moreover, in contrast to this paper, the study by Garti et al. [12] is concerned with multi-threaded applications and tries to achieve better performance by distributing application threads to different machines and executing them concurrently.

4 Local Adaptation Strategy

This section describes the main contribution of this paper, which is a local adaptation algorithm that optimises runtime object topology, through the clustering of application objects and placement of object clusters to nodes, in response to changing environmental conditions.

```

do {
    maxScore = 0.5
    maxObject = null, maxNode = null
    for each mobile object o in local node do
        for each remote node n do
            score = evaluate(o, n)
            if (score > maxScore) then
                maxScore = score
                maxObject = o
                maxNode = n
            end if
        end for
    end for
    if (maxScore > 0.5) then
        move maxObject to maxNode
    end if
while (maxScore > 0.5)

```

Fig. 1. Local Adaptation Algorithm, Basic Flow of Control

4.1 Basic Algorithm

At the abstract level, the local adaptation algorithm operates according to Fig. 1 in which individual nodes move objects to other hosts when criteria related to efficiency (performance versus resource utilisation) [23] are met. Although the basic working loop is similar to that presented in [7], the scoring function evaluate(), is significantly more capable in terms of the following: Firstly, this algorithm allows multiple efficiency sub-attributes and their inter-relationships to be considered in a single pass. Secondly, the algorithm allows specific sub-attributes to be prioritised through weighting (e.g. response time and thus performance could be favoured over network utilisation). Finally, the algorithm allows the specification of the extent to which a certain attribute should be favoured over others. For example, should performance be increased by a small amount given a high cost in resource utilisation?

The algorithm evaluates, using metric-based models [7], possible migration options based on the available local mobile objects and remote nodes. This can be used for ranking purposes, or for selecting the highest score provided it is greater than a predetermined threshold, in order to establish which object migration to carry out. The algorithm stops when the highest score no longer exceeds the threshold or when there are no more local mobile objects.

An explanation of how the sub-algorithm ‘evaluate’ produces its scores is given in the following section.

4.2 Using Metrics to Calculate Decision Making Scores

There are a number of possible general approaches to decision making based on multiple attributes, which differ in terms of how they specify criteria for the decision making process. These include linear [24] and non-linear [25] approaches which can be specified for the general case (independent of efficiency) according to equations 1 and 2 respectively.

$$S = (W_1 I_1 + W_2 I_2 + \dots + W_m I_m) \quad (1)$$

In order to evaluate such functions, and thus produce a decision making score (S) that can be used to rank and execute actions, the level of satisfaction of the individual indicators (I_i) must be calculated. This is done by normalising the values to the unitary interval ($0 \leq I_i \leq 1$) where: 0.5 means the indicator just equals its satisfaction criterion; > 0.5 means that as the value of the indicator increases towards 1, the greater it satisfies the individual criterion up to the maximum level of satisfaction of 1, corresponding to the maximum measurable value for the metric associated with the indicator. Conversely, < 0.5 means that as the indicator value decreases towards 0, the less it satisfies the criterion down to the minimum level of satisfaction of 0, corresponding to the minimum possible value for the associated metric.

Furthermore, both the linear and non linear variations of the aggregate decision making function include weights (W_i), to represent the relative importance of the individual indicators when calculating the decision making score S . A further requirement of both functions is that $(W_1 + W_2 + \dots + W_m) = 1$, where $W_i \geq 0$ for $i = 1 \dots m$.

Equation 2 is a non-linear ‘weighted power mean’ [25], which in addition to allowing the specification of relative importance via weights, also allows the specification of whether an indicator is mandatory, alternative, or neutral.

$$S(r) = (W_1 I_1^r + W_2 I_2^r + \dots + W_m I_m^r)^{1/r} \quad (2)$$

where $-\infty \leq r \leq +\infty$, $S(-\infty) = \min(I_1, I_2, \dots, I_m)$ and $S(+\infty) = \max(I_1, I_2, \dots, I_m)$.

The power r is a real number parameter selected to achieve the desired indicator relationship of the aggregation function. Equation 2 is equivalent to equation 1 when $r = 1$, which models the neutrality relationship where all indicators are considered equally with significance attributed only to their value and weighting. Equation 2 is supra-additive for $r > 1$, which models indicator replaceability (or disjunction) meaning that one or more higher indicators are favoured at the cost of lower indicators. Alternatively, it is sub-additive for $r < 1$ (with $r \neq 0$), thereby modelling indicator simultaneity (or conjunction), thus favouring the situation where there are no low indicators.

For example, consider the following situation where $S = (0.5 I_1^r + 0.5 I_2^r)^{1/r}$. Table 1 shows the effect of r on S for different values of I_1 and I_2 . Firstly, when $r = 1$, S is the average of I_1 and I_2 . Secondly, when $r > 1$, S is greater than the average (i.e. closer to the highest indicator I_2). Finally, when $r < 1$, S is less than the average (i.e. closer to the lowest indicator I_1).

The ‘weighted power mean’ approach can be specialised with the efficiency sub-attributes (indicators) resource utilisation and response time, to model decision

Table 1. Effect of the parameter r on the scoring model S

r	I_1	I_2	S	I_1	I_2	S	I_1	I_2	S
10	0.5	0.5	0.5	0.4	0.6	0.56	0.1	0.9	0.84
1	0.5	0.5	0.5	0.4	0.6	0.50	0.1	0.9	0.50
-10	0.5	0.5	0.5	0.4	0.6	0.43	0.1	0.9	0.11

making and thus facilitate adaptation based on efficiency (Equations 3 and 4). These specific equations serve as the computational basis for the sub-algorithm ‘evaluate’ in Fig. 1 and the empirical studies of the adaptation behaviour of the local adaptation algorithm, which are presented in the following section. It should be noted that similar equations could be specified for other quality attributes (and their sub-attributes) as has been done in the case of web applications [26]. This is however beyond the scope of this paper and thus quality attributes such as reliability, which could be relevant to local adaptation, are left as the subject of future work.

$$S_E = (W_{MU} I_{MU}^r + W_{NU} I_{NU}^r + W_{PU} I_{PU}^r + W_{RT} I_{RT}^r)^{1/r} \quad (3)$$

$$I_i = 0.5 + 0.5 (d - k) / (2 \times max) \quad (4)$$

where $d = ru_d$ or rt_d , $k = ru_k$ or rt_k (see Equations 5 and 6 in the Appendix), and $max = ru_{max}$ or rt_{max} .

5 Empirical Evaluation

A series of empirical studies were conducted by deploying a prototype of a Taxi Dispatching System (TDS) on the MobJeX framework [6], in order to evaluate the adaptation algorithm presented in the previous section. The TDS application was chosen because it is simple enough to be described within this paper, but complex enough in terms of its design, functionality and object topology, to provide meaningful evaluation of the metric-based adaptation strategy presented in the previous section. Furthermore, the TDS application has sufficient scope to suggest explicit directions for future work. Note that the TDS application design is based on five main objects (a location manager [lm], client manager [cm], job manager [jm], taxi manager [tm], and user interface [ui]). A description of TDS in terms of its functional requirements and main use cases appears in [7].

Software metrics were collected offline via a static analysis of the TDS source code, whereas resource utilisation and performance metrics were obtained online during execution via the resource monitor component of MobJeX. Furthermore, the actual adaptation decisions produced by the local adaptation algorithm of section 0 were carried out using MobJeX to transparently (i.e. without impacting application state) migrate objects between nodes at runtime. The experimentation was conducted in a research laboratory with a 100 Mbps Ethernet network, which was isolated from the rest of the university to eliminate the confounding effect of external traffic. Furthermore, since the TDS application was relatively small with four main mobile objects¹ (and a non-mobile object: [ui]), we used a small machine cluster with three identical nodes (1 GHz, 512MB, Windows XP) and used software to cap the network bandwidth at 11 Mbps to emulate a slower 802.11b wireless network.

Note that it is not the intention of this paper to test the efficiency or accuracy of the metrics collection process itself but rather the effectiveness of the local adaptation algorithm when fed appropriate metrics. It is the subject of future work to look at the

¹ These main objects were effectively object clusters since they held references to a number of smaller worker objects that were also moved as part of the migration process of MobJeX.

impact of the actual metrics collection process and how this must be factored into the decision making process when deciding if a given object topology is more efficient than another. Furthermore, a working implementation of this strategy will require the existence of some protocol among the nodes to guarantee the execution of the adaptation at only one node at any given time. This together with careful selection of parameters will suffice in most cases to prevent undesirable side effects such as system thrashing and ‘pinball’ migration (i.e. an object keeps migrating between two or more nodes).

Three separate experiments, each testing a number of variations of the tuning parameters, were conducted in order to test the main characteristics of the local adaptation algorithm. These experiments involved adaptation in response to changes in processor, memory and network utilisation respectively, with varying prioritisation of performance versus resource utilisation. In addition, different tuning parameters for the algorithm, in terms of attribute weights and values of ‘r’ (see equation 2 in section 0) were chosen in order to test the impact of parameter choices.

For each of the experiments the following efficiency metrics were collected: 1) Performance in terms of average scenario response time in milliseconds; 2) The standard deviation of processor utilisation across nodes 3) The standard deviation of memory utilisation across nodes 4) The standard deviation of network utilisation across nodes. Note that the standard deviation measurements of 2-4 served as an indication of resource utilisation in terms of load balance, where a lower standard deviation represented a more even balance across nodes. Additionally, all the resource utilisation measurements were presented as percentages, derived from the ratio usage/capacity of resources, for a single node (see Table 6 in the appendix).

These four measurements serve as dependent variables for the experiments and were collected at three different stages: 1) Before a change in resource utilisation has occurred (Initial State); 2) After an event has been triggered to signify a change in the utilisation of a resource, where no adaptation has been performed (No-Adaptation Final State); and 3) After the same event, but where the local adaptation algorithm has been executed to optimise the object topology of the application in response to the change in resource utilisation (Adaptation Final State). This data allows us to directly measure the effectiveness of the local adaptation algorithm in terms of maintaining efficiency in a dynamically changing environment.

5.1 Adapting to Changes in Processor Utilisation

The experiment in this section evaluates the impact of the adaptation strategy on efficiency when processor utilisation changes. This was done using the following linear ($r = 1$) scoring model that applies equal weights to performance and processor utilisation, whilst ignoring the other indicators:

$$S = (0 I_{MU} + 0 I_{NU} + 0.5 I_{PU} + 0.5 I_{RT}), \text{ with } r = 1.$$

Measurements were taken based on the following three states, with thresholds $rt_k = 1\text{ms}$ and $ru_k = 1\%$ used to invoke the maximum adaptation outcome.

- *Initial State (IS):* TDS executing centrally using only node [Z].
- *Event:* The processor load in node [Z] increases significantly (up to 90% utilisation).

- *Adaptation Final State (AFS)*: TDS executing in a distributed manner using nodes [X], [Y] and [Z].
- *No-Adaptation Final State (NFS)*: the same as IS.

Expected outcome. After the event where processor utilisation of node [Z] increases to 90% utilisation, the adapted application should: 1) Perform better than the non-adapted state due to the extra processor time available, and 2) Processor usage should be more balanced, thus resulting in improved overall efficiency.

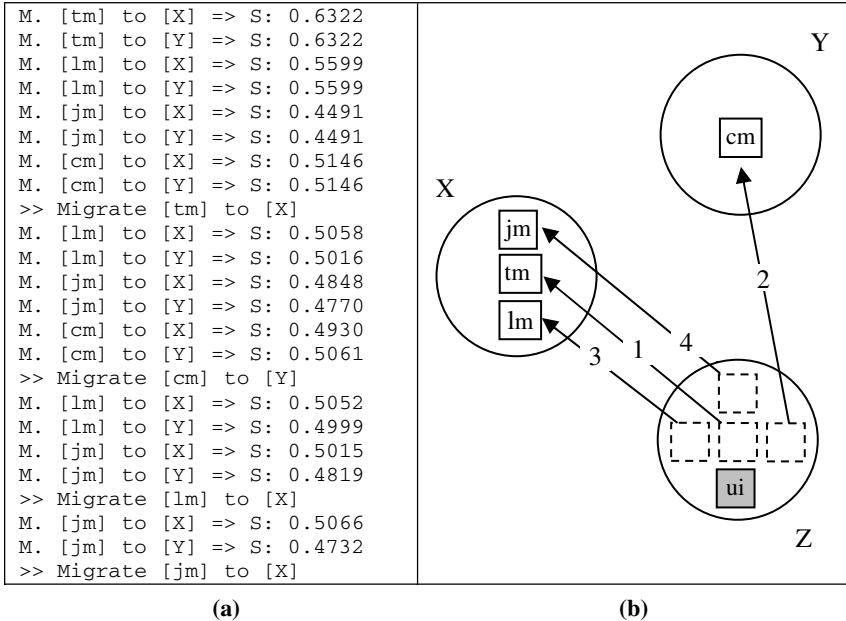


Fig. 2. Algorithm trace and the corresponding schematic diagram for scoring model S

Algorithm Trace. Fig. 2(a) shows the complete algorithm trace for model S and Fig. 2(b) illustrates the adaptation (migration) process schematically.

In the first decision, [tm] and [lm] get high I_{RT} because their methods are characterised by a high Number of Executed Instructions (NEI , see Appendix) and a low Size of Serialised Parameters (SSP). This implies a low invocation time (IT) and thus the overhead of a remote invocation becomes less significant since the Execution Time (ET), which does not change significantly upon migration, is the main contributor to overall Response Time (RT) since $RT = ET + IT$. However, [tm] gets a higher I_{PU} than [lm], and hence a higher decision score S , due to the higher Number of Invocations (NI) of its methods.

At the time of the second decision, following the first migration, node [Z] is less loaded. Therefore, moving further objects affects performance negatively ($\downarrow I_{RT}$). Consequently, objects such as [cm] with high NEI but low NI score higher than objects such as [lm] since they do not significantly affect performance but do improve

processor load balance ($\uparrow I_{PU}$). [cm] is not coupled to [tm], so locating [cm] in a different node ([Y]) does not add remote invocations (which would affect performance) but does improve load distribution.

In the third decision, moving [lm] to node [X] does not negatively affect performance due to remote invocations because [lm] is only coupled to [tm], which resides in node [X], thus slightly improving processor load distribution.

Finally, in the last decision, moving [jm] to node [X] leaves performance at the same level, since it is mostly coupled to [tm], but slightly improves processor utilisation, by scoring just above the decision making threshold of 1%.

Results. Table 2 shows the efficiency metrics for the 3 states described in this section. As expected, after the environmental change and subsequent adaptation, the execution of the adapted application performs considerably better than the non-adapted application and the processor usage is more balanced, thus resulting in improved overall efficiency.

Table 2. Efficiency metrics for scoring model S

	IS	AFS	NFS
Avg. SRT	13.78 ms	24.76 ms	40.09 ms
Std. Dev. PU	37.70 %	43.45 %	51.61 %

5.2 Adapting to Changes in Memory Utilisation

The experiments in this section evaluate the impact of the adaptation strategy on efficiency attributes, when memory utilisation changes. Therefore, in the scoring models of this section different weights were assigned to the memory utilisation and response time indicators, while the weight of network and processor utilisation remained fixed at zero. The algorithm was tested with various tuning parameters in terms of indicator weights and values of the parameter r as described by the following five variations: 1) Neutral model with equal weights; 2) Neutral model with performance favoured over memory utilisation; 3) Neutral model with memory utilisation favoured over performance; 4) Disjunctive model; and 5) Conjunctive model. The following memory utilisation event and three states apply to all five variations of the scoring function.

- *Initial State (IS):* TDS executing centrally using only node (Z).
- *Event:* The amount of free memory in node [Z] decreases significantly (by 256 MB).
- *Adaptation Final State (AFS):* TDS executing in a distributed manner using nodes [X] and [Z].
- *No-Adaptation Final State (NFS):* the same as IS.

In all cases the thresholds $rt_k = 1$ ms and $ru_k = 1\%$ were used to invoke the maximum adaptation outcome.

Neutral model with equal weights. $S_i = (0.5 I_{MU} + 0 I_{NU} + 0 I_{PU} + 0.5 I_{RT})$, $r = 1$. This scoring model tests how the algorithm reacts when the same weight is assigned to

performance and memory utilisation, and the other indicators are ignored (value of 0), for the linear case of $r = 1$.

Expected outcome. After the event where memory utilisation of node [Z] increases by 256MB, the adapted application should perform better than the non-adapted state due to less paging activity, and the memory usage should be more balanced, thus resulting in improved overall efficiency.

Algorithm Trace. Fig. 3 shows a partial algorithm trace for model S_1 . Note that for brevity, the final stages of the trace, where no migration decisions occur ($S_1 < 0.5$), are omitted. The fact that [cm] gets the highest score can be explained in two parts. Firstly, [cm] is characterised by the highest Object Memory Size (OMS, see Appendix), therefore, the decision to move this object implies a larger μ_{mu} , which in turn implies a larger I_{mu} (see equation 4). Furthermore, the methods of [cm] are characterised by a low Number of Invocations (NI) and thus moving this object does not imply a low r_{rt} , thereby avoiding a low I_{rt} . Note that the performance indicator I_{rt} is considered in the context of overall Scenario Response Times (SRT), following a chain of method calls, rather than the individual response time of a single method. Consequently, the low number of invocations has a minimal effect on the overall performance of the application scenario to which the method calls belong. Finally, although another object [lm], scores slightly better for I_{rt} , it does not balance the load as much as [cm], therefore scoring a lower I_{mu} and thus a lower overall score S_1 .

```

Move [tm] to [X] => S: 0.4938
Move [tm] to [Y] => S: 0.4938
Move [lm] to [X] => S: 0.5248
Move [lm] to [Y] => S: 0.5248
Move [jm] to [X] => S: 0.3388
Move [jm] to [Y] => S: 0.3388
Move [cm] to [X] => S: 0.5266
Move [cm] to [Y] => S: 0.5266
>> Migrate [cm] to [X]
...
>> No Migration

```

Table 3. Efficiency metrics (scoring model S_1)

	IS	AFS	NFS
Avg. SRT	19.57 ms	26.97 ms	43.09 ms
Std. Dev. MU	9.14 %	34.12 %	37.54 %

Fig. 3. Algorithm trace for model S_1

Results. Table 3 shows the efficiency metrics for the 3 states described above. As expected, after the environmental change and subsequent adaptation, the execution of the adapted application performs considerably better than the non-adapted application and the memory usage is marginally more balanced, hence resulting in improved overall efficiency.

Neutral model with performance outweighing memory utilisation

$S_2 = (0.4 I_{\text{MU}} + 0 I_{\text{NU}} + 0 I_{\text{PU}} + 0.6 I_{\text{RT}})$, $r = 1$. This scoring model tests how the algorithm reacts when different weights are assigned to the indicators of interest, with performance considered more important than memory utilisation, again for the neutral (linear) case where $r = 1$.

Expected outcome. This should be similar to the previous case; however performance should be higher than S_1 , possibly at the expense of memory load balance across nodes.

Algorithm Trace. Fig. 4. shows the partial algorithm trace for model S_2 . Since performance is more important (i.e. has a higher weight) than memory usage, objects with a high performance indicator I_{RT} score higher overall for S_2 because of the additional weighting compared with I_{MU} . In this case, [lm] gets the highest I_{RT} because although its methods are characterised by a high Number of Executed Instructions (NEI), the low Size of Serialised Parameters (SSP) implies a lower invocation time (IT) and thus the overhead of a remote invocation becomes less significant since the Execution Time (ET), which does not change upon migration, is the main contributor to overall Response Time (RT) since $RT = ET + IT$.

Results. Table 4 shows the efficiency metrics for the 3 application states. Again, overall efficiency is improved in the adapted versus non-adapted state, however as predicted, the performance gain is greater and the memory load less balanced than model S_1 .

```
M. [tm] to [X] => S: 0.4895
M. [tm] to [Y] => S: 0.4895
M. [lm] to [X] => S: 0.5290
M. [lm] to [Y] => S: 0.5290
M. [jm] to [X] => S: 0.3071
M. [jm] to [Y] => S: 0.3071
M. [cm] to [X] => S: 0.5288
M. [cm] to [Y] => S: 0.5288
>> Migrate [lm] to [X]
...
>> No Migration
```

```
M. [tm] to [X] => S: 0.5152
M. [tm] to [Y] => S: 0.5152
M. [lm] to [X] => S: 0.5040
M. [lm] to [Y] => S: 0.5040
M. [jm] to [X] => S: 0.4974
M. [jm] to [Y] => S: 0.4974
M. [cm] to [X] => S: 0.5154
M. [cm] to [Y] => S: 0.5154
>> Migrate [cm] to [X]
...
>> No Migration
```

Fig. 4. Algorithm trace for scoring model S_2

Fig. 5. Algorithm trace for scoring model S_3

Table 4. Efficiency metrics (scoring model S_2)

	IS	AFS	NFS
Avg. SRT	19.57 ms	23.90 ms	43.09 ms
Std. Dev. MU	9.14 %	36.10 %	37.54 %

Neutral model with memory utilisation outweighing performance

$S_3 = (1 I_{MU} + 0 I_{NU} + 0 I_{PU} + 0 I_{RT})$, $r = 1$. This scoring model is similar to S_2 , however in this case memory utilisation is completely favoured over performance, with a weight of one and zero respectively.

Expected Outcome. After the event, the memory utilisation of the adapted execution should be more balanced than the non-adapted execution, possibly at the expense of performance.

Algorithm Trace. The algorithm trace for model S_3 can be seen in Fig. 5. Since memory utilisation is more important than performance, objects with a high I_{MU} score

a greater value for S_3 than objects such as [lm] which have a high I_{RT} but lower I_{MU} . [cm] gets the highest I_{MU} for the same reason as S_1 . However, although the indicator values change, the adaptation decision remains the same as S_1 .

Results. The results are not repeated since the migration decision, and thus the results, are the same as S_1 (Table 3). This occurred because although the different weighting affected the indicator scores, the size of the objects compared with the memory capacity (MC) of the nodes, and the size of the memory utilisation event (256MB), was not significant enough to exceed the threshold of 1% after the initial case of moving [cm].

$$\text{Disjunctive Model. } S_4 = (0.5 I_{MU}^r + 0 I_{NU}^r + 0 I_{PU}^r + 0.5 I_{RT}^r)^{1/r}, r = 8.$$

Expected outcome. Although the indicators of performance and memory usage are equally important, setting $r > 1$ means that a big improvement on one attribute, even at the cost of deterioration of another, is preferred over a small or medium improvement on both attributes. Therefore, after the event, adapted execution should perform either significantly better than the non-adapted execution or the memory usage should be considerably more balanced.

Algorithm Trace. The algorithm trace for model S_4 can be seen in Fig. 6. The decision to move [lm] achieves the highest value for a relevant indicator, and thus its associated decision score S_4 is also higher.

```
M. [tm] to [X] => S: 0.4970
M. [tm] to [Y] => S: 0.4970
M. [lm] to [X] => S: 0.5276
M. [lm] to [Y] => S: 0.5276
M. [jm] to [X] => S: 0.4562
M. [jm] to [Y] => S: 0.4562
M. [cm] to [X] => S: 0.5274
M. [cm] to [Y] => S: 0.5274
>> Migrate [lm] to [X]
...
>> No Migration
```

```
M. [tm] to [X] => S: 0.4757
M. [tm] to [Y] => S: 0.4757
M. [lm] to [X] => S: 0.5075
M. [lm] to [Y] => S: 0.5075
M. [jm] to [X] => S: 0.1814
M. [jm] to [Y] => S: 0.1814
M. [cm] to [X] => S: 0.5189
M. [cm] to [Y] => S: 0.5189
>> Migrate [cm] to [X]
...
>> No Migration
```

Fig. 6. Algorithm trace for scoring model S_4

Fig. 7. Algorithm trace for scoring model S_5

Results. Again, the results are not repeated since the migration decision, is the same as S_2 (Table 4). This is in line with the expected outcome since the highest score is produced by the migration option with the highest indicator (I_{RT}).

$$\text{Conjunctive model. } S_5 = (0.5 I_{MU}^r + 0 I_{NU}^r + 0 I_{PU}^r + 0.5 I_{RT}^r)^{1/r}, r = -100.$$

Expected outcome. Although the indicators of performance and memory usage are equally important, setting $r < 1$ produces a lower score if any of the indicators is low regardless of whether any of the other attributes have a high value. Hence, a small or medium improvement on both attributes is preferred over a big improvement on one

at the cost of deterioration of another. Therefore, the adapted application should be more efficient overall than the non-adapted application.

Algorithm Trace. Fig. 7 illustrates the algorithm trace for model S_5 . The decision to move [cm] achieves the highest minimum value for a relevant indicator, and thus its associated decision score S_5 is also higher.

Results. Here the value of $r = -100$ was chosen since it demonstrated that even an extremely low value of r did not change the result from the default case of $r = 1$, since this case had already chosen the migration option that produced the result with the highest minimum indicator. As such, the scenario results, which were the same as the linear case for S_1 in Table 3, were according to expectation.

5.3 Adapting to Changes in Network Utilisation

This final scenario assigns different weights to the network utilisation and response time indicators, while fixing the weights of memory and processor utilisation at zero, in order to test the ability of the local adaptation algorithm to respond to changes in network utilisation. As in the previous sub-section, the algorithm was tested with various tuning parameters in terms of indicator weights and values of the parameter r as follows:

- $S'_1 = (0.5 I_{RT}^r + 0 I_{PU}^r + 0 I_{MU}^r + 0.5 I_{NU}^r)^{1/r}$, $r = 1$
- $S'_2 = (0.1 I_{RT}^r + 0 I_{PU}^r + 0 I_{MU}^r + 0.9 I_{NU}^r)^{1/r}$, $r = 1$
- $S'_3 = (0.9 I_{RT}^r + 0 I_{PU}^r + 0 I_{MU}^r + 0.1 I_{NU}^r)^{1/r}$, $r = 1$
- $S'_4 = (0.5 I_{RT}^r + 0 I_{PU}^r + 0 I_{MU}^r + 0.5 I_{NU}^r)^{1/r}$, $r = 100$
- $S'_5 = (0.5 I_{RT}^r + 0 I_{PU}^r + 0 I_{MU}^r + 0.5 I_{NU}^r)^{1/r}$, $r = -100$

As in the previous cases the thresholds $rt_k = 1$ ms and $ru_k = 1\%$ were used to invoke the maximum adaptation outcome. The following three states apply to all five variations based on a network utilisation event:

- *Initial State (IS):* TDS executing in a distributed manner using nodes [X] and [Z] as in AFS of the previous section.
- *Event:* Increased network utilisation causes the network bandwidth available to node [X] to decrease significantly to 0.55 Mbps.
- *Adaptation Final State (AFS):* TDS executing in a distributed manner using nodes [Y] and [Z].
- *No-Adaptation Final State (NFS):* the same as IS.

Fig. 8 shows the complete algorithm trace for model S'_1 , while Table 5 shows the efficiency metrics for the three states described above. As expected, after the environmental change and subsequent adaptation, the execution of the adapted application performs better than the non-adapted application, and in fact even better than the initial execution, while network usage becomes more balanced, resulting in improved overall efficiency.

Note that in this scenario it was not possible to change the outcome of the algorithm by varying either the weights or the parameter r , since the initial state provides the algorithm with only two migration options, as can be seen in Fig. 9.

Since the option to move [cm] to [Y] scores better for both of the efficiency indicators I_{NU} and I_{RT} , the decision scores S'_1 to S'_5 were always higher for the decision to move [cm] to [Z] and thus tuning the algorithm based on different values did not have an effect on the outcome in this case. This is not problematic since this scenario has both demonstrated the ability of the adaptation algorithm to improve efficiency in response to a change in network utilisation, and shown that where there is a clearly preferential decision, the weighting and tuning parameters will not necessarily affect the adaptation outcome.

Table 5. Efficiency metrics for model S'

	IS	AFS	NFS
Avg. SRT	39.34 ms	38.09 ms	43.35 ms
Std. Dev. NU	1.86 %	3.12 %	36.90 %

M. [cm] to [Y] => S: 0.5786
M. [cm] to [Z] => S: 0.5468
>> Migrate [cm] to [Y]

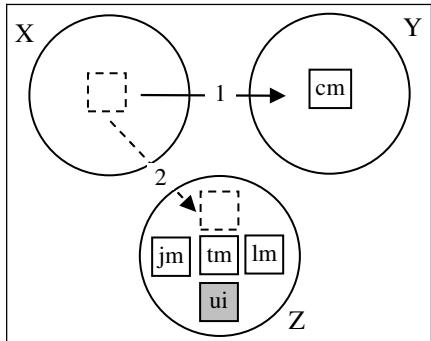


Fig. 8. Algorithm trace for model S'

Fig. 9. Schematic diagram for model S'

6 Conclusions

This paper has proposed and empirically evaluated a local adaptation strategy for distributed mobile applications, which is based on a series of formal models and a suite of metrics for mobile applications introduced by the authors in a recent paper.

The experiments demonstrated a potential practical application of the local adaptation algorithm using a number of distinct scenarios involving runtime changes in processor, memory and network utilisation. Improvements in the efficiency of the adapted distributed application were demonstrated, in comparison with the non adapted state, in response to these changes in resource utilisation. The adaptation process itself involved the rearrangement of object topology, achieved by dynamically moving objects between nodes, based on metrics that characterised: 1) The software components of the application in terms of attributes such as method invocation overhead, method intensity, size of serialised parameters, serialised object size etc. 2) Efficiency in terms of performance (response time), and resource utilisation attributes representing the percentage utilisation of specific resources such as processor, network or memory; and 3) Algorithm tuning parameters allowing preference to be given to specific attributes and their indicators, as well as the decision to favour higher attribute values over lower ones.

The authors believe that the application of the local adaptation algorithm, as well as the new set of metrics upon which it is based, are both novel and show significant potential thus serving as a significant contribution. Nevertheless, there are a number

of limitations of the present study, as well as related topics that are beyond the scope of this paper, which can serve as the basis for ongoing work.

Firstly, although this paper studies a real application running on an existing framework for distributed mobile applications, future work could look at a larger example application using an increased number of nodes and a more diverse range of adaptation scenarios.

Secondly, the overhead of the metrics collection and evaluation process was not explicitly considered and thus although it was demonstrated that the adaptation algorithm could produce favourable results, future work will examine the optimisation of the collection and evaluation process so as to not offset the gains achieved through adaptation via object mobility. This will be done by integrating the metrics collection strategies and the local adaptation algorithm into the MobJeX framework, thus providing a test bed for further optimisation of this process.

Thirdly, while this paper has considered efficiency in terms of performance, and memory, network, and processor utilisation; there are other efficiency factors that could be considered such as physical storage, and financial cost where for example, contrary to a fixed local area network, a 3G or wireless network could incur a per data unit charge. Furthermore, power consumption, especially in terms of battery usage in mobile devices, is another important area for future study since the decision to favour the utilisation of the previously considered resources can have a direct impact on power consumption [27].

Finally, it should be noted that similar metrics and indicators could be specified for other quality attributes (and their sub-attributes) such as reliability [23], in which case adaptation could be performed based on more than one quality attribute.

Acknowledgements

This work is part of the Applications Program of the Australian Telecommunications Cooperative Research Centre: <http://www.atcrc.com/>.

References

1. Emmerich, W., *Engineering Distributed Objects*: Wiley. (2000).
2. Sun Microsystems. *Java 2 Micro Edition*. URL: <http://java.sun.com/j2me/>. [May 2005]
3. Symbian Ltd. *Symbian OS*. URL: <http://www.symbian.com/>. [May 2005]
4. Microsoft Corporation. *.NET Compact Framework*. URL: <http://msdn.microsoft.com/mobile/prodtechinfo/devtools/netcf/>. [May 2005]
5. Ryan, C. and S. Perry, *Client/Server Configuration in a Next Generation Internet Environment: End-User, Developer, and Service Provider Perspectives*. In Proceedings: 2003 Australian Telecommunications, Networks and Applications Conference (ATNAC). Melbourne, Australia. (2003)
6. Ryan, C. and C. Westhorpe, *Application Adaptation through Transparent and Portable Object Mobility in Java*. In Proceedings: CoopIS/DOA/ODBASE (LNCS 3291). Larnaca, Cyprus: Springer-Verlag. p. 1262-1284 (2004)
7. Ryan, C. and P. Rossi, *Software, Performance and Resource Utilisation Metrics for Context-Aware Mobile Applications*. In Proceedings: International Software Metrics Symposium. Como, Italy: IEEE Computer Society. (2005)

8. Segarra, M. and F. Andre, *A Framework for Dynamic Adaptation in Wireless Environments*. In Proceedings: Technology of Object-Oriented Languages and Systems: IEEE. p. 336-347 (2000)
9. Aziz, B. and C. Jensen, *Adaptability in CORBA: The Mobile Proxy Approach*. In Proceedings: International Symposium on Distributed Objects and Applications: IEEE Computer Society. p. 295-304 (2000)
10. Fox, A., et al., *Adapting to network and client variation using active proxies: Lessons and perspectives*. IEEE Personal Communications. **5**(4): p. 10-19 (1998)
11. Noble, B., *System Support for Adaptive, Mobile Applications*. IEEE Personal Communications. **7**(1): p. 44-49 (2000)
12. Garti, D., et al., *Object Mobility for Performance Improvements of Parallel Java Applications*. Parallel and Distributed Computing. **60**(10): p. 1311-1324 (2000)
13. Ben-Shaul, I., et al., *Dynamic Self Adaptation in Distributed Systems*. In Proceedings: Self-Adaptive Software: First International Workshop. Oxford: Springer. p. 134-142 (2000)
14. Blair, G.S., et al., *A principled approach to supporting adaptation in distributed mobile environments*. In Proceedings: Software Engineering for Parallel and Distributed Systems. International Symposium on: IEEE. p. 3-12 (2000)
15. Capra, L., W. Emmerich, and C. Mascolo, *CARISMA: context-aware reflective middleware system for mobile applications*. Software Engineering, IEEE Transactions on. **29**(10): p. 929-945 (2003)
16. Chang, F. and V. Karamcheti, *Automatic configuration and run-time adaptation of distributed applications*. In Proceedings: Ninth IEEE International Symposium on High Performance Distributed Computing. Pittsburg, Pennsylvania. p. 11-20 (2000)
17. Moura, A., et al., *Dynamic support for distributed auto-adaptive applications*. In Proceedings: Workshop on Aspect Oriented Programming for Distributed Computing Systems. Vienna, Austria: IEEE. p. 451-456 (2002)
18. Silva, F., M. Endler, and F. Kon, *Developing Adaptive Distributed Applications: A Framework Overview and Experimental Results*. In Proceedings: CoopIS/DOA/ODBASE (LNCS 2888): Springer. p. 1275 - 1291 (2003)
19. Vanegas, R., et al., *QuO's runtime support for quality of service in distributed objects*. In Proceedings: International Conference on Distributed Systems Platforms and Open Distributed Processing. The Lake District, England: Springer. p. 207-224 (1998)
20. Venkatasubramanian, N., C. Talcott, and G. Agha, *A Formal Model for Reasoning About Adaptive QoS-Enabled Middleware*. ACM Transactions on Software Engineering and Methodology. **13**(1): p. 86-147 (2004)
21. Maia, R., R. Cerqueira, and N. Rodriguez, *An Infrastructure for Development of Dynamically Adaptable Distributed Components*. In Proceedings: CoopIS, DOA, and ODBASE (LNCS 3290). Larnaca, Cyprus: Springer-Verlag. p. 1285-1302 (2004)
22. Jing, J., A. Helal, and A. Elmagarmid, *Client-Server Computing in Mobile Environments*. ACM Computing Surveys. **31**(2): p. 118-157 (1999)
23. ISO/IEC, *Information Technology - Software Product Quality - Part 1: Quality Model*. 2001, International Standards Organisation: Geneva.
24. Gilb, T., *Software Metrics*. Massachusetts: Winthrop. (1977).
25. Dujmovic, J., *A Method for Evaluation and Selection of Complex Hardware and Software Systems*. In Proceedings: International Conf. on Resource Management and Performance Evaluation of Enterprise Computer Systems. Turnersville, N.J. p. 368-378 (1996)
26. Olsina, L. and G. Rossi, *Measuring Web Applications Quality with WebQEM*. IEEE Multimedia. **9**(4): p. 20-29 (2002)
27. Chen, G., et al., *Studying energy trade offs in offloading computation/compilation in Java-enabled mobile devices*. Parallel and Distributed Systems, IEEE Transactions on. **15**(9): p. 795-809 (2004)

Appendix

Table 6. Summary of the Relationships among Metrics

Attribute	Metric	Unit	Related to
Code	Object Compilation Volume	Executable Code Size (ECS)	byte MCT, NU
	Object Serialisation Volume	Serialised Object Size (SOS)	byte MIT, NU
	Object Memory Volume	Object Memory Size (OMS)	byte MU
	Method Execution Volume	Execution Memory Size (EMS)	byte MU
	Method Body Intensity	Number of Executed Instructions (NEI)	int ET, PU
	Method Interface Volume	Size of Serialised Parameters (SSP)	byte IT, NU
	Method Invocation Frequency	Number of Invocations (NI)	int NU, PU
Efficiency	Method Execution Cost	Method Execution Time (ET)	ms -
	Method Invocation Cost	Method Invocation Time (IT)	ms -
	Object Migration Cost	Migrate Instance Time (MIT)	ms -
	Class Migration Cost	Migrate Class Time (MCT)	ms -
	Network Utilisation	Network Usage (NU)	byte IT, MCT, MIT
	Memory Utilisation	Memory Usage (MU)	byte -
	Processor Utilisation	Processor Usage (PU)	int/s ET

$$rt_d = \sum_{i=1..m} [\mathbf{NI}_i * (rt_i^C - rt_i^D)] - mot, \text{ with } mot = \mathbf{MCT} + \mathbf{MIT} \text{ and } rt = \mathbf{IT} + \mathbf{ET} \quad (5)$$

$$ru_d = \left| \frac{ru^C}{rc^C} - \frac{ru^D}{rc^D} \right| - \left| \frac{ru^C - ru^O}{rc^C} - \frac{ru^D + ru^O}{rc^D} \right|, \text{ with } ru^O = ru^F + \frac{\sum_{i=1..m} (ru_i * \mathbf{NI}_i)}{m} \quad (6)$$

where $mu^F = \mathbf{OMS}$, $mu_i = \mathbf{EMS}$, $nu^F = \mathbf{SOS} + \mathbf{ECS}$, $nu_i = \mathbf{SSP}$, $pu^F = 0$, $pu_i = f(\mathbf{NEI})$, rc^C and rc^D = resource capacity of the current and destination nodes of the object respectively, and m = number of methods [7].

Middleware for Distributed Context-Aware Systems^{*}

Karen Henricksen¹, Jadwiga Indulska²,
Ted McFadden¹, and Sasitharan Balasubramaniam²

¹ CRC for Enterprise Distributed Systems Technology (DSTC)
karen@itee.uq.edu.au, mcfadden@dstc.edu.au

² School of Information Technology and Electrical Engineering,
The University of Queensland
jaga@itee.uq.edu.au, sasib@tssg.org

Abstract. Context-aware systems represent extremely complex and heterogeneous distributed systems, composed of sensors, actuators, application components, and a variety of context processing components that manage the flow of context information between the sensors/actuators and applications. The need for middleware to seamlessly bind these components together is well recognised. Numerous attempts to build middleware or infrastructure for context-aware systems have been made, but these have provided only partial solutions; for instance, most have not adequately addressed issues such as mobility, fault tolerance or privacy. One of the goals of this paper is to provide an analysis of the requirements of a middleware for context-aware systems, drawing from both traditional distributed system goals and our experiences with developing context-aware applications. The paper also provides a critical review of several middleware solutions, followed by a comprehensive discussion of our own PACE middleware. Finally, it provides a comparison of our solution with the previous work, highlighting both the advantages of our middleware and important topics for future research.

1 Introduction

The proliferation of standalone and embedded computing devices in our work and home environments, combined with a variety of networking technologies, increases the importance of context-awareness in distributed applications. Context-aware applications adapt to changes in the environment and user requirements. This dynamic adaptation provides the degree of autonomy needed to free users from the current computer-centric model of human-computer interaction. For example, sensor-based “smart home” applications can unobtrusively support elderly people in everyday tasks, such as remembering to take medications or providing early detection of behavioural changes.

* The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government’s CRC Programme (Department of Education, Science, and Training).

The complexity of developing context-aware applications makes middleware an essential requirement. The middleware solutions proposed so far for context-aware systems address basic issues traditionally addressed by middleware for distributed systems, including paradigms for coordination and communication between distributed components. They also offer support for gathering and managing context information, in order to simplify application development and promote sharing of context information and context sensing components. However, many additional requirements are not met. For instance, most solutions do not adequately support the deployment and configuration of new components, the dynamic reconfiguration of components, or user privacy.

In this paper, we evaluate the current state-of-the-art in middleware for distributed context-aware applications, including the middleware developed in our PACE (Pervasive, Autonomic, Context-aware Environments) project. Based on the evaluation, we also highlight a set of open research problems in this area.

The structure of the paper is as follows. In Sections 2 and 3, we characterise context-aware systems and introduce a set of requirements for middleware for these systems. In Section 4, we review a set of middleware solutions and analyse them with respect to the requirements. In Sections 5 and 6, we introduce our PACE middleware and demonstrate how the middleware is used to support the development of a context-aware vertical handover application. Finally, in Sections 7 and 8, we provide an analysis of our solution, a discussion of open research challenges, and a summary of the contributions of this paper.

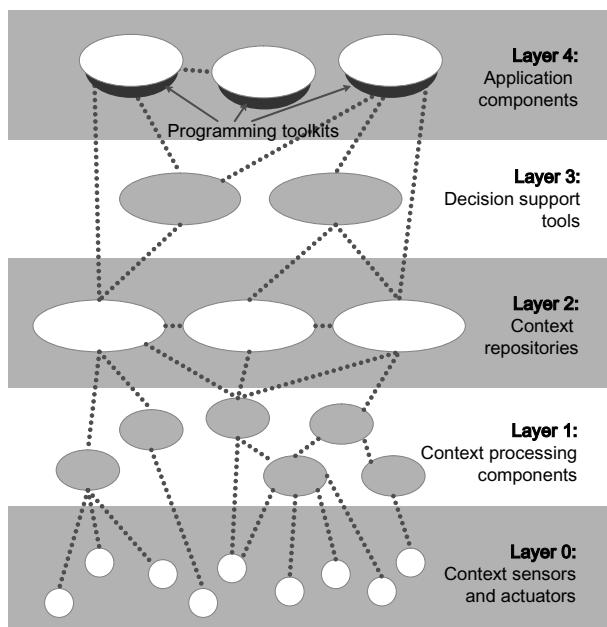


Fig. 1. Components of a context-aware system

2 Characteristics of Context-Aware Systems

Context-aware systems consist of a variety of distributed components. Early systems were relatively simple, and were often constructed simply as distributed application components communicating directly with local or remote sensors. Today, it is widely acknowledged that additional infrastructural components are desirable, in order to reduce the complexity of context-aware applications, improve maintainability, and promote reuse. Figure 1 illustrates the distributed components that can be found in many current context-aware systems. In addition to application components, sensors and actuators, shown at the two extremities in this layered model, these systems include:

- components that (i) assist with processing sensor outputs to produce context information that can be used by applications and (ii) map update operations on the higher-order information back down to actions on actuators (layer 1);
- context repositories that provide persistent storage of context information and advanced query facilities (layer 2); and
- decision support tools that help applications to select appropriate actions and adaptations based on the available context information (layer 3).

Programming toolkits are often also incorporated at the application layer (layer 4) to support the interactions of the application components with other components of the context-aware system.

3 Middleware Requirements

In this paper, we refer to the components that reside between the layer 4 application components and the layer 0 sensors and actuators - together with the communications framework that binds the distributed components together - as middleware for context-aware systems. This middleware must address many of the requirements of traditional distributed systems, such as heterogeneity, mobility, scalability, and tolerance for component failures and disconnections. In addition, it must protect users' personal information, such as location and preferences, in accordance with their privacy preferences, and ensure that automatic actions taken by context-aware applications on behalf of users can be adequately understood and controlled by users. Finally, the large number of distributed components that are present in context-aware systems introduces a requirement for straightforward techniques for deploying, configuring and managing networks of sensors, actuators, context processing components, context repositories, and so on. A detailed summary of these requirements is provided in Table 1.

4 A Survey of Middleware for Context-Aware Systems

In this section, we review and analyse some of the proposed middleware solutions for context-aware systems. We focus on solutions that span multiple layers of

Table 1. Requirements for middleware for context-aware systems

1. Support for heterogeneity	Hardware components ranging from resource-poor sensors, actuators and mobile client devices to high-performance servers must be supported, as must a variety of networking interfaces and programming languages. Legacy components may be present.
2. Support for mobility	All components (especially sensors and applications) can be mobile, and the communication protocols that underpin the system must therefore support appropriately flexible forms of routing. Context information may need to migrate with context-aware components. Flexible component discovery mechanisms are required.
3. Scalability	Context processing components and communication protocols must perform adequately in systems ranging from few to many sensors, actuators and application components. Similarly, they must scale to many administrative domains.
4. Support for privacy	Flows of context information between the distributed components of a context-aware system must be controlled according to users' privacy needs and expectations.
5. Traceability and control	The state of the system components and information flows between components should be open to inspection - and, where relevant, manipulation - in order to provide adequate understanding and control of the system to users, and to facilitate debugging.
6. Tolerance for component failures	Sensors and other components are likely to fail in the ordinary operation of a context-aware system. Disconnections may also occur. The system must continue operation, without requiring excessive resources to detect and handle failures.
7. Ease of deployment and configuration	The distributed hardware and software components of a context-aware system must be easily deployed and configured to meet user and environmental requirements, potentially by non-experts (for example, in "smart home" environments).

the system architecture shown in Fig. 1; that is, we exclude single layer solutions such as context servers (layer 2) and models for context interpretation (layer 1). We also exclude solutions that are not general, such as those that deal only with location sensing and management.

4.1 The Context Toolkit

Dey et al.'s Context Toolkit [1] provides a set of abstractions that can be used to implement reusable software components for context sensing and interpretation. The context *widget* abstraction represents a component that is responsible for acquiring context information directly from a sensor. Widgets can be combined with *interpreters*, which transform low-level information into higher-level information that is more useful to applications, and *aggregators*, which group related

context information together in a single component. Finally, *services* can be used by context-aware applications to invoke actions using actuators, and *discoverers* can be used by applications to locate suitable widgets, interpreters, aggregators and services.

The toolkit is implemented as a set of Java objects that represent the abstractions described above. These provide a basic communication protocol based on HTTP and XML. The use of these Web standards allows for interoperation with components implemented in other languages, thereby providing basic support for heterogeneity. The toolkit's *discoverers* address component discovery, which is one of the requirements for mobility. However, the toolkit does not specifically address scalability, privacy, traceability/control¹, component failures, or deployment/configuration.

4.2 Context Fusion Networks

Chen et al. [3] propose the use of *Context Fusion Networks* (CFNs) to provide data fusion services (aggregation and interpretation of sensor data) to context-aware applications. CFNs are based on an operator graph model, in which context processing is specified by application developers in terms of sources, sinks and channels. In this model, sensors are represented by sources, and applications by sinks. Operators, which are responsible for data processing, act as both sources and sinks.

Chen et al. have implemented the CFN model in the form of *Solar*, a scalable peer-to-peer (P2P) platform which instantiates the operator graphs at runtime on behalf of context-aware applications. The Solar hosts (*Planets*) support application and sensor mobility by buffering events during periods of disconnection; they also address component failures by providing monitoring and recovery, as well as preservation of component states. However, Solar does not yet address heterogeneity, privacy, or monitoring and control of the system by users.

4.3 The Context Fabric

Unlike the previous two solutions, the Context Fabric (Confab) proposed by Hong and Landay [4] is primarily concerned with privacy rather than with context sensing and processing. Confab provides an architecture for privacy-sensitive systems, as well as a set of privacy mechanisms that can be used by application developers. The architecture structures context information into *infospaces*, which store tuples about a given entity. Infospaces are populated by context sources such as sensors, and queried by context-aware applications.

Hong and Landay have implemented the infospace model using Web technologies, such that infospaces are identified by URLs and tuples are exchanged in an XML format. They provide a programming model based on *in* and *out* methods for transferring tuples into and out of infospaces. Privacy can be supported by adding operators to an infospace to carry out actions when tuples enter or

¹ However, Newberger and Dey [2] did later address monitoring and control by providing an *enactor* extension to the Context Toolkit.

leave the space; for instance, operators can be used to perform access control, notify users of information disclosure, and enforce privacy tags that describe how information can be used after it flows from one infospace to another.

As Confab focuses so heavily on privacy, it does not address traditional distributed systems requirements such as mobility, scalability, component failures and deployment/configuration. However, it does partially address heterogeneity, as it builds on platform- and language-independent Web standards. It also provides privacy-related traceability and control via the operator mechanism.

4.4 Gaia

Gaia [5] is designed to facilitate the construction of applications for smart spaces, such as smart homes and meeting rooms. It consists of a set of core services and a framework for building distributed context-aware applications. Gaia's event manager service enables applications to be developed as loosely coupled components, and can provide basic fault tolerance by allowing failed event producers to be automatically replaced. Gaia's remaining four services support various forms of context-awareness, and include: (i) a context service, which allows applications to find providers for the context information they require, (ii) a presence service, which monitors the entities entering and leaving a smart space (including people as well as hardware and software components), (iii) a space repository, which maintains descriptions of hardware and software components, and (iv) a context file system, which associates files with relevant context information and dynamically constructs virtual directory hierarchies according to the current context.

As smart spaces are typically small, constrained environments, Gaia does not address scalability (however, [6] canvasses the issues involved in federating spaces into large-scale “super spaces”). Similarly, privacy is not addressed by any of the basic services, but can potentially be provided by additional services [7], while user monitoring/control is outside Gaia's scope. Heterogeneity, mobility and component configuration can all be supported by Gaia in limited forms.

4.5 Reconfigurable Context-Sensitive Middleware

Yau et al. [8] propose a Reconfigurable Context-Sensitive Middleware (RCSM) for context-aware applications. The RCSM provides application developers with a novel Interface Definition Language (IDL) that can be used to specify context requirements, including the types of context/situation that are relevant to the application, the actions to be triggered, and the timing of these actions. The IDL interfaces are compiled to produce application skeletons; these interact at runtime with the RCSM Object Request Broker (R-ORB), which manages context acquisition, and the Situation-Awareness (SA) processor, which is responsible for managing triggers.

The R-ORB provides a context manager that uses a context discovery protocol to manage registrations of local sensors and discover remote sensors. When a context-aware application starts up, the discovery protocol is used to look for local or remote sensors that satisfy the application's context requirements.

Table 2. Middleware support for the requirements of context-aware systems
 (Key: ✓ = comprehensive, √ = partial, × = none)

Requirement	Context Toolkit	CFN/ Solar	Context Fabric	Gaia	RCSM
<i>Support for heterogeneity</i>	✓	×	✓	✓	✓
<i>Support for mobility</i>	✓	√	×	✓	×
<i>Scalability</i>	×	√	×	×	×
<i>Support for privacy</i>	×	×	✓	×	×
<i>Traceability and control</i>	×	×	✓	×	×
<i>Tolerance for failures</i>	×	√	×	✓	×
<i>Ease of deployment/ configuration</i>	×	√	×	✓	✓

The prototype described by Yau et al. does not satisfy the heterogeneity requirement, as it supports only C++ applications for the Windows CE platform; however, the IDL compiler could potentially be modified to produce skeletons for a variety of platforms and communication protocols. In addition, the context discovery protocol is not flexible enough to support mobility or component failure, and Yau et al. do not attempt to address scalability, privacy or traceability/control. The main strength of the approach comes from the use of an IDL to specify context requirements. This makes it possible to incorporate new types of context and context-aware behaviour by editing and recompiling IDL interfaces, and partially addresses ease of deployment and configuration.

4.6 Analysis

Table 2 summarises the capabilities of the surveyed solutions and shows that comprehensive solutions do not yet exist. A further shortcoming, which is not revealed in the table, is that none of the solutions provide decision support (layer 3 functionality). Our own middleware, which we discuss next, introduces decision support and addresses a large subset of the requirements listed in Table 1.

5 The PACE Middleware

Our middleware was developed as part of the PACE project, which investigates a variety of issues related to pervasive computing, including the design of context-aware applications and solutions for modelling and managing context information. An early form of the middleware was presented in [9]; however, further tools and components have been added subsequently as we developed further context-aware applications and uncovered additional requirements. Our current version of the middleware consists of:

- a context management system (layer 2);
- a preference management system that provides customisable decision-support for context-aware applications (layer 3);
- a programming toolkit that facilitates interaction between application components and the context and preference management systems (layer 4); and
- tools that assist with generating components that can be used by all layers, including a flexible messaging framework.

These components and tools have been developed according to the following design principles:

1. The model(s) of context information used in a context-aware system should be explicitly represented within the system. This representation should be separate from the application components (layer 4) and the parts of the system concerned with sensing and actuation (layers 0 and 1), so that the context model can evolve independently, without requiring any components to be re-implemented.
2. The context-aware behaviour of context-aware applications should be determined, at least in part, by external specifications that can be customised by users and evolved along with the context model (again, without forcing re-implementation of any components).
3. The communication between application components, and between the components and middleware services, should not be tightly bound to the application logic, so that a significant re-implementation effort is required when the underlying transport protocols or service interfaces change.

The following sections provide an overview of the components and tools that make up the middleware. In Section 6, we illustrate their use in the development of a context-aware system that supports vertical handover of media streams.

5.1 Context Management System

In our middleware, the context management system fulfils the requirements of layer 2 as discussed in Section 2: that is, it provides aggregation and storage of context information, in addition to performing query evaluation. It uses a two-layered context modelling approach, in which context can be expressed both in terms of fine-grained facts and higher-level situations which capture logical conditions that can be *true*, *false* or *unknown* in a certain context. All information is stored in the fact representation, but can be queried by either retrieving specific facts based on template matching, or evaluating situation definitions over a set of facts. Our context modelling approach has been well documented in previous papers [9,10], and therefore is not described in detail here. However, an example fact-based context model will be shown later in Section 6.

The context management system consists of a distributed set of context repositories. Each repository manages a catalog, which is a collection of context models consisting of fact type and situation definitions. Applications may

define their own context models or share them with other applications. Context-aware components are not statically linked to a single repository, but can discover repositories dynamically by catalog name (and potentially also other attributes). Several methods of interacting with a context repository are currently permitted, in order to support a range of client programming languages and platforms; likewise, a variety of discovery mechanisms can be used, including context-based discovery, which allows for matching based on context attributes.

Each repository is capable of performing access control, although this feature can be switched off if it is not required. The access control mechanism allows users to define privacy preferences that dictate the circumstances (i.e., situations) in which context information can be queried and updated. The privacy preferences are stored and evaluated by the preference management system, which we describe in the following section.

Our current prototype consists of a context management layer running on top of a relational database management system. This is written in Java using JDBC² to query and manipulate a set of context databases³. It provides clients with the following interfaces:

- *query*: supports situation evaluation and retrieval of facts matching supplied templates;
- *update*: allows insertion, deletion and modification of facts, as well as insertion of new situation definitions;
- *transaction*: allows clients to create read-only transactions within which a sequence of queries can be executed against a consistent set of context information, regardless of concurrent updates;
- *subscription*: allows monitoring of situations and fact types, using callbacks to notification interfaces implemented by clients; and
- *metadata*: allows clients to discover the fact types and situations that are defined by models in the catalog.

In addition to invoking methods on repositories using Java RMI, clients can use a Web interface (based on XML and HTTP) or programming language stubs generated from a context model specification. The latter method can potentially accommodate arbitrary programming languages and communication protocols; currently, we generate stubs for Java and Python, using Elvin [11], a content-based message routing scheme, as the underlying communication paradigm. One of the benefits of Elvin is that it allows for complex interactions (including 1:N and N:M communication, not only 1:1 as supported by RMI and HTTP), which allows (for example) queries and updates to be simultaneously routed to multiple context repositories. We discuss the stubs further in Section 5.5.

² <http://java.sun.com/products/jdbc/>

³ Note that this is not the most efficient implementation in terms of query/update time and throughput, but we have found the performance adequate for all of the context-aware applications we have developed so far.

Currently, our context repositories behave independently of one another; however, we are developing a model for replicating context information across several repositories and allowing clients to cache their own context information for use during disconnections.

5.2 Preference Management System

A preference management system provides layer 3 functionality that builds on functionality of the context management system. It assists context-aware applications with making context-based decisions on behalf of users. Its main roles are to provide storage of user preference information and evaluation of preferences - with respect to application state variables and context information stored by the context management system - to determine which application actions are preferred by the user in the current context. Applications can connect to, and store their preference information in, one or more preference repositories.

The preferences are defined in terms of our novel preference model, which allows the description of context-dependent requirements in a form that enables them to be combined on-the-fly to support decisions about users' preferred choice(s) from a set of available candidates. For example, the preference model can be used to decide which mode of input or output should be employed for particular users according to their requirements and current contexts. A detailed description of the preference model is outside the scope of this paper, but further information can be found in earlier papers [9,10,12].

The benefits of a preference-based approach to decision-making are that customisation and evolution of context-aware behaviour can be supported in a straightforward manner; preferences can be shared and exchanged between applications; and new types of context information can be incorporated into decision-making processes simply by adding new preferences, without the need to modify the application components.

The implementation of the preference management system bears strong resemblance to that of the context management system, and therefore we discuss it relatively briefly. It provides the following interfaces:

- *update*: allows new preferences to be defined and grouped appropriately into sets (for instance, by owner and purpose);
- *query*: provides preference evaluation based on the information stored in the context management system;
- *transaction*: allows a set of preference evaluations to occur over a consistent set of context information, regardless of concurrent updates occurring within the context management layer; and
- *metadata*: allows retrieval of preference and preference set definitions.

In a similar manner to the context repositories, the preference repositories respond to requests from clients over a variety of communication protocols. However, Java clients need not interact directly with repositories; instead, they are provided with a Java programming toolkit that assists with discovery of, and interaction with, repositories. We describe the toolkit in the following section.

5.3 Programming Toolkit

The programming toolkit complements the functionality of the preference management layer by implementing a simple conceptual model for formulating and carrying out context-based choices. The model provides a mechanism for linking application actions with candidate choices. It also allows one or more of the actions to be automatically invoked on the basis of the results of evaluating the choices with respect to preference and context information, using the services of the preference and context management systems.

A significant benefit of the toolkit is that it makes the process of discovering and communicating with the preference and context management systems transparent to applications. It also helps to produce applications that are cleanly structured and decoupled from their context models, and thus better able to support changes in the available context information. These changes can result from evolution of the sensing infrastructure over time, or problems such as disconnection or migration from a sensor-rich environment to a sensor-poor one.

The toolkit is currently only implemented in Java, using RMI for communication with remote components; however, it could be ported to other programming languages and communication protocols in the future.

5.4 Messaging Framework

To facilitate remote communication between components of context-aware systems - which may be either application components or middleware services such as the ones described in Sections 5.1 and 5.2 - we provide a flexible messaging framework. In the tradition of middleware such as CORBA, the framework aims to provide various forms of transparency, such as location and migration transparency. It maps interface definitions to communication stubs that are appropriate for the deployment environment. These stubs are considerably simpler for the programmer to work with than the APIs of the underlying transport layers, and can also be automatically re-generated at a later date, allowing for substitution of transport layers without modifying the application.

Stubs can be generated for a variety of programming languages and communication protocols (including message-based and RPC-based protocols). To date, however, we have focused on producing Java and Python stubs for the Elvin publish/subscribe content-based message routing scheme. Elvin is particularly appropriate for building context-aware systems because it decouples communication from cooperation. Because it delivers messages based on matches between message content and the subscriptions of listeners, rather than based on explicit addressing, it is able to tolerate mobility, support complex interactions (not only 1:1 interactions as in the case of RPC/RMI), and allow for spontaneous interactions between components without the need for an explicit discovery/configuration step. The ability to add new listeners into the system on-the-fly is also useful for debugging and generating traces.

In the future, we plan to extend the messaging framework to other protocols appropriate for context-aware systems (for example, context-based routing schemes such as GeoCast [13], which performs routing based on location).

5.5 Schema Compiler Toolset

The final piece of our middleware is a set of tools capable of producing custom components to assist with developing and deploying context-aware systems, starting from context models specified using the two-layered context modelling approach that we briefly outlined in Section 5.1. The tools take input in the form of a textual representation of a context model (a context *schema*), perform checks to verify the integrity of the model, and produce the following outputs:

- SQL scripts to load and remove context model definitions from the relational databases used by our context repositories;
- model-specific helper classes to simplify source code concerned with carrying out context queries and updates; and
- context model interface definitions compatible with the messaging framework.

The first output simplifies the deployment and evolution of context models. By automating the mapping of context models into the database structures stored by the context repositories, errors that might arise during the hand-coding of SQL scripts or JDBC code to manipulate the repositories can be avoided. Similarly, updates to context models can be supported simply by re-generating and re-executing the scripts. In the future, we envision extending the tools to produce alternative scripts for context repositories that are not SQL-based.

The second output is designed to simplify the programming of components that query or update a context model, and includes classes that represent basic value types, fact types and situations defined by the model. By programming with these classes, rather than the generic APIs provided by the context management layer, type checking becomes possible at compile time and standard IDE features such as code completion can better be exploited.

The final output is used to produce stubs for transmitting/receiving context information over communications infrastructure such as Elvin. The context transmitters can be used by layer 0 and layer 1 components (sensors, actuators and processing components) to transmit context information to one or more context repositories. Similarly, the context receivers can be used at layer 2 to listen for context updates that require mapping to operations on context repositories.

Further information about the context schema toolset can be found in [14].

6 Case Study: Vertical Handover

We now illustrate how our middleware assists with the development of distributed context-aware systems, using a vertical handover application as a case

study. This application represents just one of the context-aware applications we have developed using the middleware; others are described in earlier papers [9,12]. The application is concerned with adapting the streaming of media to a mobile user according to the context. The adaptation occurs at the application layer rather than the network layer (e.g., using Mobile IP) because of stringent Quality of Service (QoS) requirements. The application adapts by handing over the stream, either between network interfaces on a single computing device or between interfaces on different devices. A handover can potentially occur in response to any context change; for example, the user moving into range of a network that offers higher bandwidth than the current network, or the signal strength of the current network dropping.

The handover process is managed by adaptation managers and proxies. The adaptation managers use the context management system to monitor significant context changes, in order to determine when a vertical handover should occur, and to which network interface. The proxies perform the handover process. One proxy is co-located with the transmitter, while other proxies are located within the same networks as the receivers. The transmitter's local proxy (proxy-transmitter) is responsible for redirecting the stream when it receives a handover instruction from an adaptation manager. During the handover process, the proxy-transmitter transmits the stream to both the original and the new proxy. This is referred to as doublecasting. The proxies within the receivers' networks are responsible for forwarding the streams to the receiver(s) executing on the client device(s). When the handover is complete, the proxy-transmitter stops transmitting to the original receiver proxy.

6.1 Implementation

The architecture of the system, including both application components and supporting middleware components, is shown in Fig. 2. In the remainder of this section, we demonstrate how our middleware was used to implement the system.

Context Model. The context model used by the vertical handover prototype is shown in Fig. 3. The main objects described by the model are computing devices, networks, network interfaces, streams and proxies. The model captures associations between computing devices and network interfaces, proximity between devices, mappings of streams to proxies and network interfaces, basic QoS information related to network interfaces (current signal strength and bandwidth), and other type and configuration information. Much of the information is user- or application-supplied (i.e., static or profiled in the terminology of our context modelling approach); however, proximity between devices is sensed using wireless beacons, and current network connectivity, signal strength and bandwidth are all sensed by monitors running in the network.

The context model, and its instantiation at run-time with concrete facts, is managed by a set of context repositories as shown in Fig. 2. Each local network may contain one or more repositories. In the example system architecture shown

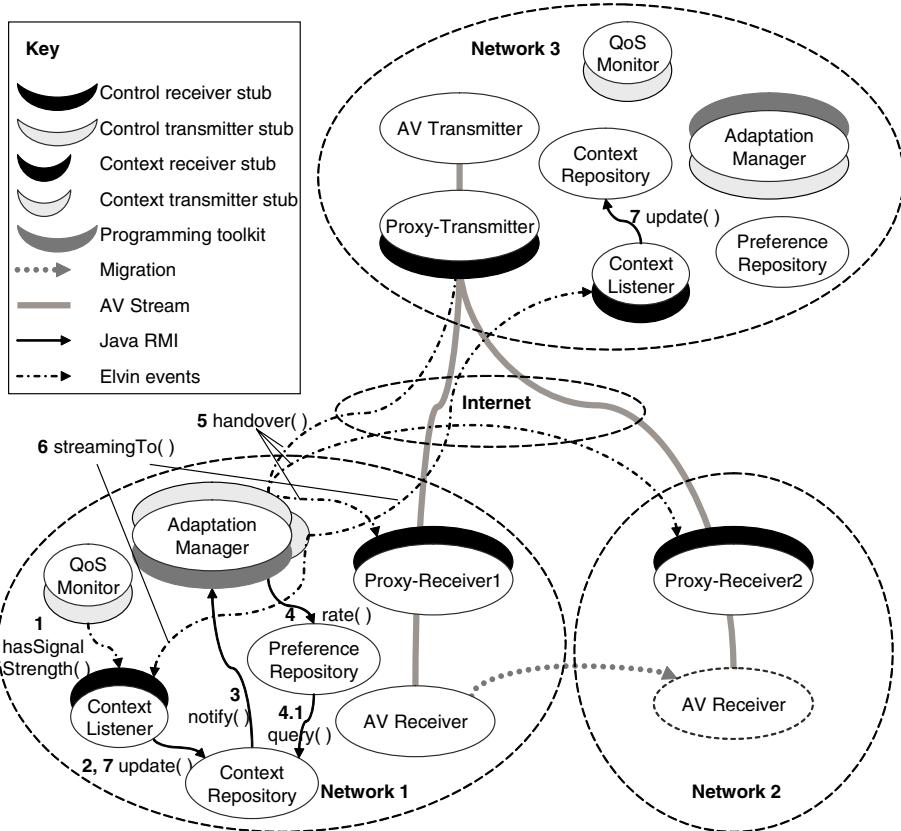


Fig. 2. Vertical handover architecture

in the diagram, two of the local networks possess their own context repositories, while the other network does not. However, the design of the system is such that many other configurations are also possible.

The schema compiler toolset described in Section 5.5 was used to map the context model to appropriate database structures when the context repositories were deployed. The toolset was also used to generate context transmitter and receiver stubs, which are used by the wireless beacons (not shown in the diagram), network monitors and adaptation managers (shown for networks 1 and 3) to report context information to the context repositories over Elvin, via context listeners that map the Elvin notifications to RMI context repository updates.

Adaptation Managers. The context-aware functionality of the application is concentrated within the adaptation managers. These are the components that are responsible for determining when handover is required, according to the current context and user preferences. Therefore, the adaptation managers are the components that interact with the context and preference repositories.

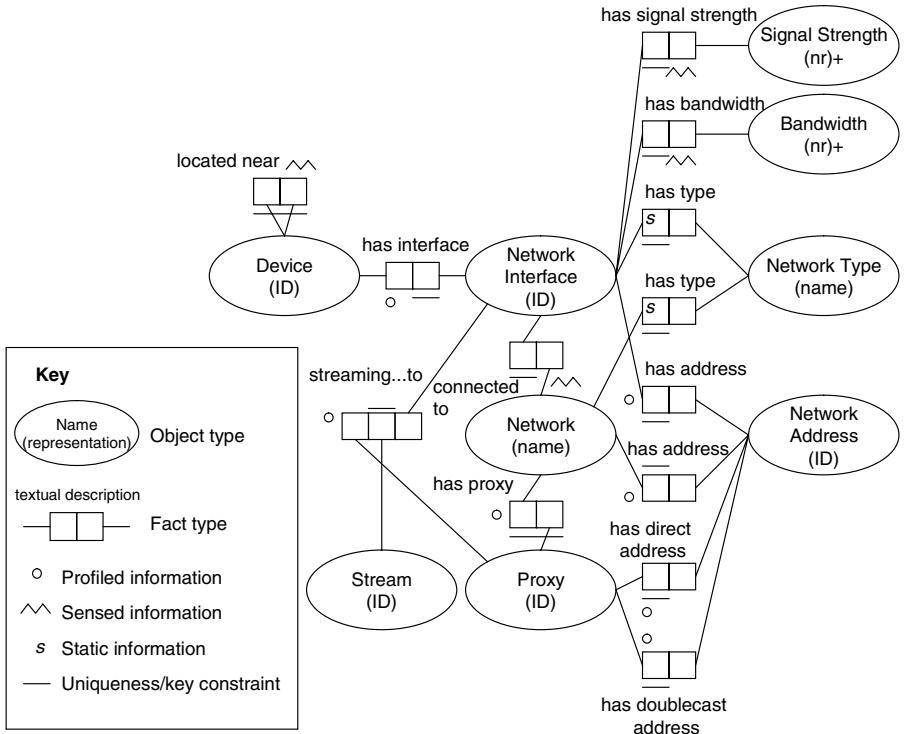


Fig. 3. The context model used in the vertical handover application

According to the design principles outlined in Section 5, the adaptation managers are not tightly coupled to the context model. The only direct interaction that occurs between the adaptation managers and the context repositories is in the form of subscriptions/notifications, which allow the managers to learn about significant context changes and report new streaming configurations. The subscriptions monitor the state of the sensed fact types. When an adaptation manager is notified of a change (for example, a drop in signal strength, as shown in Fig. 2), it uses the programming toolkit described in Section 5.3 to connect to a preference repository, re-evaluate the user's preferences, and determine whether the current network interface is still the preferred one (step 4 in the figure). The bulk of the context evaluation occurs during this step, as a side-effect of the preference evaluation (step 4.1). New context information can be easily incorporated into the evaluation simply by extending the user preferences (i.e., the implementation of the adaptation manager does not need to change).

When an adaptation manager determines that a handover is required, it communicates with the proxies (step 5) and the context listeners (step 6) using Elvin. The manager first transmits a handover instruction to the proxies using Elvin stubs produced by the PACE messaging framework described in Section 5.4. After instructing the proxies to perform the handover, the adaptation manager

updates the stream state information stored in the context repositories (i.e., the “streaming...to” fact type shown in Fig. 3), using a context transmitter stub to transmit the information to the context listeners.

7 Analysis

In this section, we briefly analyse the PACE middleware with respect to the requirements set out in Table 1 and compare it to the earlier solutions surveyed in Section 4. Based on the analysis, we also highlight some areas for future work.

Heterogeneity. As the PACE messaging framework can generate stubs for a variety of programming languages (and, in the future, transport layers), it offers strong support for heterogeneity. The PACE middleware is also capable of accommodating legacy components, such as the transmitters and receivers in the vertical handover system. Thus, PACE’s support for heterogeneity is more comprehensive than the solutions surveyed in Section 4. However, Yau et al.’s solution bears some similarities to our approach, and could be extended to generate skeletons for a variety of platforms as discussed in Section 4.5.

Mobility. The use of Elvin within the messaging framework facilitates component mobility, as demonstrated in the vertical handover system, and often removes the need for component discovery. Local context and preference repositories can be dynamically discovered by mobile context-aware components using a variety of service discovery protocols. PACE therefore provides a level of mobility support that is comparable to the best solutions surveyed in Section 4. In the future, we plan to extend PACE’s current support for mobility by introducing caching/hoarding models for context and preference information, to allow mobile components to store local copies of information that is relevant to users.

Scalability. Our current implementation of the PACE middleware does not address scalability or performance, and the same can be said of almost all of the solutions surveyed in Section 4. This is unsurprising, as all have been developed as research prototypes. As future work, we intend to develop models for federating context and preference managers across large scale systems and a large number of administrative domains.

Privacy. We address privacy by providing access control for sensitive context information. Thus, our middleware provides more privacy support than all of the surveyed solutions, with the exception of Confab. However, controlling access to context information addresses only one aspect of privacy. To address other aspects, we intend to add access control to our preference management system and combine this access control with context-based authentication [13]. Further information about our current work on privacy can be found in [15] and [16].

Traceability and Control. We showed in Section 4 that traceability and control are not addressed at all by previous middleware except in relation to privacy. The PACE middleware begins to address this problem. The use of Elvin facilitates the generation of traces, as event listeners can be added on-the-fly and

event traces can be tailored by adjusting the Elvin subscriptions. Our preference model also provides a basic mechanism for user control and customisation. In the future, we envisage opening up the service layers to clients to allow inspection (and manipulation) of context and preference evaluations. Traces of these evaluations can be selectively revealed to users to explain system behaviours.

Tolerance for Failures. Our solution's failure tolerance ranks behind that of Solar but ahead of the remaining solutions surveyed in Section 4. Although our middleware does not yet detect or repair failed components, its use of Elvin allows a loose coupling of components, minimising the impact of disconnections and failures. In addition, our context and preference models were both designed with the assumption that context information will generally be imperfect. This introduces some tolerance for failed sensors, sensing errors, and so on.

Deployment and Configuration. Finally, the PACE middleware provides more advanced support for component deployment and configuration than previous solutions. Specifically, the messaging framework simplifies the deployment of components on top of a variety of platforms, while the schema compiler toolset facilitates the deployment of new context models. However, further extensions to the middleware are needed to facilitate the scalable deployment and configuration of infrastructural components such as sensors.

8 Conclusions

In this paper, we showed that middleware is essential for building context-aware systems and introduced a list of requirements that this middleware must address. We also analysed the current state-of-the-art in the area and provided a comprehensive discussion and evaluation of our own PACE middleware. Our solution ranked the best or equal best for the majority of the requirements (heterogeneity, mobility, traceability/control and deployment/configuration), and above average for two of the remaining three requirements (privacy and tolerance for failures). A further advantage of the PACE middleware is that it provides decision support (i.e., layer 3 functionality), unlike the other solutions we surveyed. However, many problems have not yet been adequately addressed by our work or that of the broader research community - for example, scalable deployment, configuration and management of sensors, caching and hoarding of context information to support mobility, and mechanisms for revealing aspects of the system state to facilitate user understanding and control.

References

1. Dey, A.K., Salber, D., Abowd, G.D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction* **16** (2001) 97–166
2. Newberger, A., Dey, A.: Designer support for context monitoring and control. Technical Report IRB-TR-03-017, Intel Research Berkeley (2003)

3. Chen, G., Li, M., Kotz, D.: Design and implementation of a large-scale context fusion network. In: 1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous), IEEE Computer Society (2004) 246–255
4. Hong, J.I., Landay, J.A.: An architecture for privacy-sensitive ubiquitous computing. In: 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys), Boston (2004)
5. Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H., Nahrstedt, K.: Gaia: A middleware infrastructure for active spaces. *IEEE Pervasive Computing, Special Issue on Wearable Computing* **1** (2002) 74–83
6. Al-Muhtadi, J., Chetan, S., Ranganathan, A., Campbell, R.: Super spaces: A middleware for large-scale pervasive computing environments. In: Workshop on Middleware Support for Pervasive Computing (PerWare), PerCom'04 Workshop Proceedings, Orlando (2004) 198–202
7. Al-Muhtadi, J., Ranganathan, A., Campbell, R., Mickunas, M.D.: Cerberus: A context-aware security scheme for smart spaces. In: 1st IEEE International Conference on Pervasive Computing and Communications (PerCom), Fort Worth (2003) 489–496
8. Yau, S.S., Huang, D., Gong, H., Seth, S.: Development and runtime support for situation-aware application software in ubiquitous computing environments. In: 28th Annual International Computer Software and Application Conference (COMPSAC), Hong Kong (2004) 452–457
9. Henricksen, K., Indulska, J.: A software engineering framework for context-aware pervasive computing. In: 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom), IEEE Computer Society (2004) 77–86
10. Indulska, J., Henricksen, K., McFadden, T., Mascaro, P.: Towards a common context model for virtual community applications. In: 2nd International Conference on Smart Homes and Health Telematics (ICOST). Volume 14 of Assistive Technology Research Series., IOS Press (2004) 154–161
11. Segall, B., Arnold, D., Boot, J., Henderson, M., Phelps, T.: Content based routing with Elvin4. In: AUUG2K Conference, Canberra (2000)
12. McFadden, T., Henricksen, K., Indulska, J., Mascaro, P.: Applying a disciplined approach to the development of a context-aware communication application. In: 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom), IEEE Computer Society (2005) 300–306
13. Navas, J.C., Imielinski, T.: Geographic addressing and routing. In: 3rd ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), Budapest (1997)
14. McFadden, T., Henricksen, K., Indulska, J.: Automating context-aware application development. In: UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham (2004) 90–95
15. Henricksen, K., Wishart, R., McFadden, T., Indulska, J.: Extending context models for privacy in pervasive computing environments. In: 2nd International Workshop on Context Modelling and Reasoning (CoMoRea), PerCom'05 Workshop Proceedings, IEEE Computer Society (2005) 20–24
16. Wishart, R., Henricksen, K., Indulska, J.: Context obfuscation for privacy via ontological descriptions. In: 1st International Workshop on Location- and Context-Awareness. Volume 1678 of Lecture Notes in Computer Science., Springer (2005) 276–288

Timely Provisioning of Mobile Services in Critical Pervasive Environments

Filippos Papadopoulos, Apostolos Zarras, Evangelia Pitoura,
and Panos Vassiliadis

Computer Science Department, University of Ioannina, Greece
`{filip, zarras, pitoura, pvassil}@cs.uoi.gr`

Abstract. Timeliness in conventional real-time systems is addressed by employing well-known scheduling techniques that guarantee the execution of a number of tasks within certain deadlines. However, these classical scheduling techniques do not take into account basic features that characterize today's critical pervasive computing environments.

In this paper, we revisit the issue of timeliness in the context of pervasive computing environments. We propose a middleware service that addresses the timely provisioning of services, while taking into account both the mobility of the entities that constitute pervasive computing environments and the existence of multiple alternative entities, providing semantically compatible services. Specifically, we model the overall behavior of mobile entities in terms of the entities' lifetime. The lifetime of an entity is the duration for which the entity is present and available to other entities. Given a new request coming from a mobile client and a number of semantically compatible mobile entities that can fulfill the request, one of them must be selected. The proposed service realizes three different policies that facilitate the selection. With respect to the first policy, the selection is realized solely on the basis of the client's and the server's lifetimes. The second policy additionally considers the load of each server towards selecting the one that guarantees to serve the new request within the lifetime of both the client and the server. The third policy further deals with periodic service requests.

1 Introduction

Recently, the rapid emergence of WiFi and Bluetooth networks, along with the increasing computing and communication capabilities of mobile devices such as PDAs, Pocket PCs, Smart Phones and wireless-enabled laptops, foster the development of a variety of new applications towards the realization of the overall idea of pervasive computing. Enterprises facilitate their activities for their mobile employees. Airports, railway stations, cafes and shopping centers deploy wireless networks to serve their customers. The evolution of the aforementioned technologies further enables the realization of applications that can be employed to handle certain critical situations like accidents, natural catastrophes, war situations, etc.

Both daily and critical applications are characterized by the following main features:

- They consist of a set of mobile entities, providing a number of services that can be requested by other mobile entities.
- More than one mobile entity may provide semantically equivalent services.

Critical applications are further characterized by the need for *timely provisioning* of services from mobile entities to other mobile entities. Service requests come along with specific *deadlines* that should be met by the mobile entities that serve those requests. Timeliness in conventional real-time systems is addressed by employing well-known scheduling techniques such as the earliest deadline first (EDF) and the rate-monotonic scheduling (RM) [1]. These techniques guarantee that the execution of a number of tasks will take place within the required deadlines. However, the classical scheduling techniques do not take into account basic features of pervasive computing environments.

In this paper, we revisit the issue of timeliness in the context of pervasive computing environments. Specifically, *we propose techniques that address the timely provisioning of services, while taking into account (i) the mobility of the entities that constitute pervasive computing environments and (ii) the existence of multiple alternative entities, providing semantically compatible services.*

The behavior of mobile entities may be rather complicated and depends on several factors [2]. For instance, so far, there has been work towards estimating the physical motion of mobile entities [3]. An entity may become inaccessible by moving into areas that do not belong in the transmission range of a particular wireless network. Another important feature that distinguishes mobile entities from the basic building blocks of conventional distributed systems is their limited resources. For example, the limited battery of a mobile entity may render the entity permanently or temporarily inaccessible. Moreover, the entity may explicitly disable its communication or computation capabilities towards the economization of power, or because of the reception of orders from some external authority.

To facilitate the timely provisioning of services in pervasive computing environments we employ a generic notion for modeling the behavior of mobile entities. This notion shall serve as input to the scheduling techniques that we propose in this paper. Specifically, *we assume that the overall behavior of mobile entities is modeled in terms of the entities' lifetime*. The *lifetime* of an entity is defined as *the time interval during which the entity is available to other entities*. The lifetime is generic enough and can be evaluated using a combination of different means, reflecting various characteristics such as the entity's physical motion and the entity's available resources. The evaluation of the entities' lifetimes is transparent to the proposed scheduling techniques; it is a responsibility of the entities themselves as it depends on their specificities and could not be part of a middleware infrastructure that is going to be used in different kinds of critical situations. It is important to note that the lifetime of the mobile entities is actually the contract between the entities and the scheduling techniques. As long as the lifetime is given as input to the scheduling techniques, it should be respected

by the mobile entities (i.e. the entities should not become unavailable earlier). Such a demand may seem quite restrictive for any arbitrary ad-hoc community of mobile entities. However, the proposed approach is aimed at communities that have real-time requirements and it is natural to restrict their arbitrary behavior in order to satisfy them.

The existence of more than one alternative services also plays an important role towards the timely service provisioning in pervasive computing environments. This sort of redundancy must be considered in a systematic way. Before issuing a service request to a mobile entity that shall serve it, the different alternatives must be evaluated so as to select the mobile entity that may possibly guarantee correct service provisioning.

Considering the above, *in this paper we propose three different policies that enable the timely execution of mobile services in the context of critical pervasive computing environments*. The proposed policies are realized in the core of a middleware service that is incorporated within every mobile entity. Specifically, given a new request coming from a mobile client and a number of semantically compatible servers that can fulfill the request, one of them must be selected.

The ultimate goal of the selection process is to guarantee that a response will be sent back to the client within the client's lifetime.

The first of the proposed policies takes into account solely the lifetimes of the client and the server entities; it guarantees that a reply will be sent to the client as long as the server manages to serve the client's request. The second policy provides stronger guarantees by additionally examining the load of available servers towards selecting the one that can serve the new request within the lifetime of both the client and the server. The third policy further deals with periodic service requests. To deal with such cases we *extend classical real-time scheduling techniques to the needs of pervasive computing environments*. Each different policy provides different levels of timeliness in the execution of mobile services and requires different amount of resources. In this particular paper, we concentrate on the *number of messages* required in each policy since communication between mobile devices is amongst the key causes of wasting battery.

The remainder of this paper is structured as follows. Section 2 presents a motivating example, employed throughout this paper to demonstrate the use of the proposed service. Section 3 presents the overall architecture of the proposed middleware service. Section 4 details the three alternative policies we propose. Section 5 discusses related work and finally Section 6 concludes this paper with a summary of our contribution and future research issues.

2 Motivating Example

In this section we present a scenario where the timely provision of services is essential to confront a critical situation. This scenario serves to exemplify the use of the proposed middleware service. Specifically, we face the case of an accident in a nuclear plant. The plant consists of a number of different laboratories shown in Figure 1. The accident caused a rapid increment in the overall level of

radioactivity observed in the plant area. The exact radioactivity measures range from laboratory to laboratory, depending on the physical location of each one of them (i.e., the radioactivity measures are higher in labs that are closer to the area where the accident took place). The accident took place at daytime. Hence, several employees may be trapped within the different labs. Several rescue squads enter the plant area towards dealing with this situation. Each squad is in charge of a different lab and tries to locate trapped employees. Each squad consists of firemen equipped with wireless-enabled radioactivity sensors with limited processing capabilities. Communication between different squads is feasible only through the squads leaders who are additionally equipped with small laptops serving as base stations for the networks formed in each room.

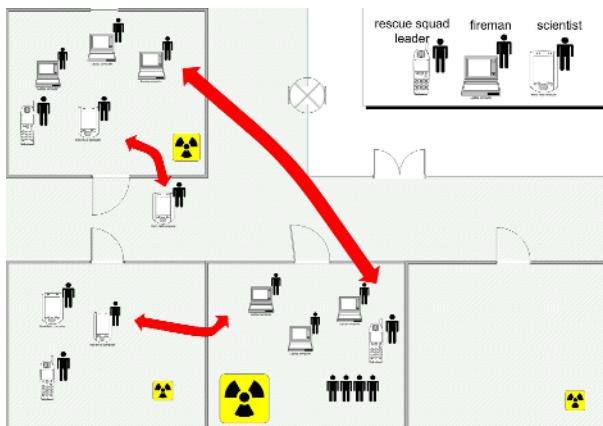


Fig. 1. Overview of the nuclear accident situation

A group of scientists also enters the accident area. The main goal of the the scientists is to gather radioactivity measures from different labs and use them for the post-mortem analysis of the situation, which shall result in estimating the impact of the accident in the plant territory. Each scientist is assigned to a different lab and carries a PDA, used for contacting the sensor-equipped firemen. Firemen and scientists can not remain in the accident area for long. Each one of them has a strict time-to-leave and within this *deadline* he must accomplish his assigned tasks. Taking for instance the firemen who accept radioactivity measure requests by the scientists, it is important to accept these requests only if they can be served within their time-to-leave and the time-to-leave of the scientists. Failing to serve these requests on-time shall delay the accurate estimation of the impact of the accident, which is critical for the identification of the particular strategy that should be followed to rapidly deal with the accident's consequences. Depending on the availability of replacements, the leaving person may be substituted by new ones. The firemen may also move from lab to lab, depending on the current situation in each one of them. For instance, a fireman may be

asked by his leader (who is constantly in contact with other leaders) to move to another lab where there exist injured employees. Locally, the leader may assign several tasks to his firemen.

In our example, the members of the rescue squads and the scientists are mobile entities. In particular, the firemen are service providers used by the scientists and the leaders of the rescue groups, which constitute the mobile clients in our critical situation.

3 Service Architecture

The overall architecture of the proposed middleware service is designed over WSAMI [4]. WSAMI is a lightweight platform developed at INRIA, which aims at facilitating the development of *ad-hoc* communities of mobile entities. WSAMI entities may execute on either stationary or mobile devices. They may provide or use a number of WEB services, conforming to the standard WEB services architecture [5]. Specifically, WSAMI services are specified using a declarative language that extends the features of the standard WEB Service Description Language (WSDL), with additional features that prescribe qualitative properties of the services such as security and transactions. Communication with the services is realized through the exchange of messages, whose format conforms with the Simple Object Access Protocol (SOAP). The WSAMI platform comprises two main subsystems used for the realization of the proposed middleware service: (1) The *CSOAP* broker, which facilitates the exchange of SOAP messages between resource constrained mobile entities; and (2) the *Naming and Discovery* (ND) service, which allows mobile entities to gather information regarding WEB services provided by other available entities.

WSAMI is a highly scalable platform since the realization of the ND service is completely distributed. Every WSAMI entity comprises an instance of the ND service, which periodically checks the environment for other instances of ND services hosted by neighboring WSAMI entities. This task is realized using the standard Service Location Protocol (SLP). The resulted information is kept locally by the service and is used afterwards for the discovery of WEB services provided by the neighboring entities.

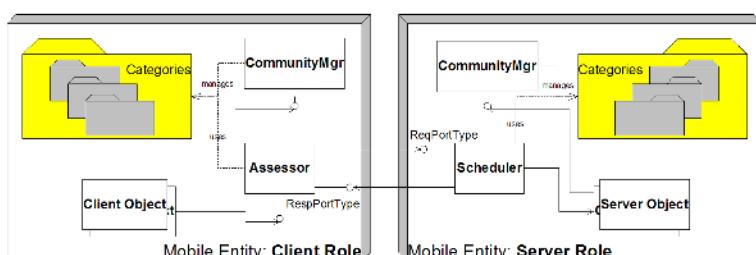


Fig. 2. Overview of the service architecture

Figure 2 gives an overview of the main components that constitute the architecture of the proposed service. A *pervasive computing environment* is a community of WSAMI entities. Each entity may play the role of a client to other mobile entities, playing the role of the server. The entity can be accessed through the use of the WEB services it provides. The mobile entity further includes a *community directory* that contains a local view of the community that corresponds to the entity. Specifically, the directory contains information regarding the WEB services that are provided by other community members, *which can be accessed by the mobile entity*. This information is divided into different *categories* depending on the different types of community members (e.g., scientists, firemen, etc.). The directory is managed by the *community manager* service. Whenever a mobile entity joins a pervasive computing environment, the community manager populates its local directory. This takes place as follows: First, the mobile entity queries the community manager for available WEB services belonging to the particular categories that interest the querying entity; the manager forwards the entity's request to ND, which subsequently contacts all other neighboring NDs; the results are collected and stored in the community directory; following, the entity configures the community manager to *periodically refresh* the directory by following the three steps mentioned above; alternatively, the entity may also *explicitly refresh* the directory.

The community manager is rather typical and is not further detailed in this paper. On the other hand, the behavior of the rest of the components showed in Figure 2 is actually the one that facilitates the timely execution of services among community members. Briefly, each mobile entity comprises an *assessor* and a *scheduler* service. The assessor service accepts as input requests from client objects encapsulated in the mobile client. For every request, the category of community entities that can serve it is further specified. Given this information, the assessor takes charge of selecting a particular mobile entity from the local community directory. Following, the assessor forwards the client request to the scheduler service of the selected entity. In particular, the scheduler accepts input messages, which are subsequently stored in a message queue maintained by the scheduler. The selection process relies on the lifetime of the community entities, which is actually part of the WSAMI description of the mobile entities. The selection process may follow three alternative policies, providing different timeliness guarantees. The three policies are the focus of the remainder of this paper.

Regarding our motivating example, each member of the rescue squads contains an instance of the server-side architecture shown in the right part of Figure 2. On the other hand, the scientists contain an instance of the client-side architecture given in the left part of the figure.

4 Timeliness Policies for the Provision of Services

Before getting into details regarding the policies of the proposed middleware service, let us formally define our execution environment. As already discussed,

a critical pervasive computing environment is a community of mobile entities, $E = \{m : MobileEntity\}$. In the most general case, a mobile entity may play both the client and the server role. Therefore, the mobile entity is a tuple $m = (D, C, S, A, lifetime)$, where D is the community directory, C is the directory manager, S is the scheduler, A is the assessor and $lifetime$ is the lifetime of the entity. Regarding the directory D we have: $D = \{ct : Category\}$ and $\forall ct \in D, ct = \{epa : EndPointAddressInfo\}$. Every element within a category contains information regarding the endpoint address of a scheduler service, provided by a neighboring mobile entity. This information comprises the URI of the service and the lifetime of the neighboring entity, $\forall epa \in ct, epa = (uri, lifetime)$.

The ultimate goal of the three policies is to guarantee that a client will receive a reply to a request made on a server entity, before leaving the community. Hereafter, we assume that the worst case communication delay for sending a SOAP message between two entities can be estimated. This estimation depends on the length of the message that is to be sent and the characteristics of the underlying network protocol. The length of the requests and responses exchanged between two entities are known to the assessor and the scheduler services. Regarding the underlying network protocol, the proposed service relies on IEEE 802.11 for wireless LANs [6]. This particular protocol provides two fundamental mechanisms for accessing the medium. The first one is called Distributed Coordination Function (DCF) and handles the retransmission of collided packets with respect to binary exponential back-off rules. The handling of packet collisions with DCF renders the estimation of the medium access delay difficult (i.e. the time required to obtain access to the medium). Consequently, the estimation of the overall delay for sending a message to a target endpoint is also complicated. The second mechanism that is provided by IEEE 802.11 is called Point Coordination Function (PCF). This mechanism guarantees collision free and time bounded packet transmissions. However, it requires the existence of a Point Coordinator (PC) that resides on an access point ¹. The PC periodically gives the right to transmit messages to each of the mobile entities that constitute a community. During this period, the mobile entities may transmit their messages or part of their messages, depending on the message length. Based on the above, the PCF mechanism is more suitable for the purpose of the proposed service.

4.1 Lifetime Policy

The first of the policies of the proposed service is based solely on the lifetimes of the mobile entities. The schedulers of the mobile servers maintain FIFO queues of requests scheduled according to the classical Round Robin (RR) algorithm.

According to the Lifetime policy, every request req is issued by a client object of a mobile entity m_{client} to the assessor service of this entity, $m_{client}.A$. The assessor must select a mobile server out of a category $ct \in m_{client}.D$ that contains

¹ The time-bounded election or substitution of a PC is an interesting issue that is complementary to our approach and is not further discussed in this paper.

information about all the alternative mobile servers that may possibly serve req , and forward req to the selected server. Within ct there may exist servers whose lifetimes are greater than the lifetime of the client and servers whose lifetimes are smaller or equal to the lifetime of the client. Selecting one of the former implies that the request may be served after the end of the client's lifetime. On the other hand, selecting one of the latter implies that the request may be served earlier than the end of the client's lifetime. In both cases, a request may not be served at all, depending on the load of the server, which is not known to the assessor. Considering the above, if there exist one or more servers whose lifetimes are smaller or equal to the client's lifetime, one of them is selected randomly by the assessor. In the opposite case, the assessor selects randomly a server with a longer lifetime. More precisely, let c_{rep} be the worst case communication delay for sending a reply message to req . Let $ct' \cup ct'' = ct$ be two disjoint subsets of ct defined as follows:

$$ct' = \{epa_{m_{server}} \in ct | epa_{m_{server}}.lifetime \leq m_{client}.lifetime - c_{rep}\}$$

$$ct'' = ct - ct'$$

Then, for the selected server m_{server} we have:

$$\begin{cases} \text{if } ct' \neq \emptyset \text{ then } m_{server} \in ct' \\ \text{else } m_{server} \in ct'' \end{cases}$$

Hence, if $m_{server} \in ct'$ the Lifetime policy guarantees that *the client will receive a reply on time, as long as the request is served*. However, there is absolutely no guarantee that the request will be served at all. The Lifetime policy is quite simple since it does not introduce any communication overhead apart from the one needed to exchange the request and the reply messages. It requires minimal information regarding the behavior of the different mobile entities of the environment.

After the selection of m_{server} , the request is forwarded towards the selected entity. Eventually, req is received by the scheduler service of m_{server} and it is placed in the request queue maintained by this service.

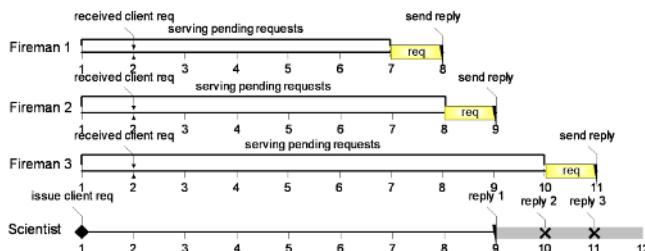


Fig. 3. Applying the Lifetime policy

Getting to our case study example, assume that a scientist enters a lab in the plant area and wants to request from a fireman the current value of radioactivity. In the same lab there exist 3 possible firemen and the scientist is supposed to

select one of them and issue his request. Suppose that the lifetimes of the scientist and the firemen are respectively 9, 8, 9 and 11 time units (Figure 3)². The communication overhead is at most 1 time unit. According to the Lifetime policy, the scientist selects Fireman 1. If the fireman manages to serve the request, in the worst case this will happen at time = 8. Consequently, the scientist will receive a reply at time = 9. On the other hand, if the scientist selects the second fireman, in the worst case his request will be completed at time = 9 and the reply will arrive at time = 10, which is too late for the scientist. Similarly, if the scientist selects the third fireman, in the worst case his request will finish at time = 9 and the reply will arrive at time = 11, which is also too late.

4.2 LifetimeLoad Policy

The second policy of the proposed service provides stronger timeliness guarantees. Still, the scheduler services of the mobile servers manage FIFO queues of requests, which are scheduled according to the classical Round Robin (RR) algorithm. However, with this policy we examine both the lifetimes and the current load of the mobile entities that may serve a client request.

As with the Lifetime policy, a request req is issued by a client object of a mobile entity m_{client} to the assessor service $m_{client}.A$. The assessor forwards directly req to the scheduler service of an accessible mobile entity, m_{server} , which is randomly selected. The scheduler service eventually receives req and examines the feasibility of its execution based on the load and the lifetime of m_{server} .

Specifically, let $N_{m_{server}}$ be the total number of pending requests for m_{server} , including req . C_{req_i} denotes the time units required for serving each pending request req_i , issued by m_{client_i} . Moreover, c_{rep_i} denotes the worst-case communication delay required to send a response back to m_{client_i} . Given that the scheduler queue is FIFO, req is the $N_{m_{server}}$ -th pending request. Every new request added to the scheduler queue introduces an additional overhead in the execution of all the other pending requests. This is due to the fact that requests are scheduled in a RR fashion. Hence, *before adding a new request* in the scheduler queue we have to verify that this additional overhead shall not delay the rest of the pending requests too long, making it impossible to send the corresponding replies back to the clients that issued the requests. Moreover, we have to verify that the new request will be served within the server's lifetime and a reply will be sent back to the client within the client's lifetime. To achieve the previous, the scheduler performs the following:

1. For every request $req_i, i = 1, \dots, N_{m_{server}}$ (including the new request, $req_{N_{m_{server}}}$), the scheduler calculates the overall time D_{req_i} required for serving it.
2. Then, to accept the new request the scheduler must verify that the following constrain holds:

$$\forall req_i, i = 1, \dots, N_{m_{server}} | D_{req_i} \leq m_{client_i}.lifetime - c_{rep_i}$$

² Note that each one of the Figures 3, 4, 5 examines three different scenarios that correspond to the selection of Fireman 1, 2 and 3 respectively.

Upon the verification of the above constraint the scheduler service reports back to the client assessor. If the constraint holds the report is positive and the scheduler continues serving pending requests including req . On the opposite case, the report is negative and the scheduler forgets req . Eventually, the assessor of m_{client} receives the report from m_{server} . In case the report is negative, another mobile entity is selected and the same procedure is followed. If the reports of all the available mobile entities are negative, req is dropped by m_{client} .

Calculating D_{req_i} is realized as follows. Given that req_i is in the i -th position of the FIFO queue and that serving it relies on RR, it takes i time units for req_i to be placed at the end of the queue. Let $Q = \{A_{req_k} | k = 1, \dots, N'_{m_{server}}\}$ be the remaining time units required for the execution of each one of the pending requests at the time when req_i will be placed at the end of the queue. Note that the cardinality of Q may be less than $N'_{m_{server}}$ given that there may be requests from the original queue that will be completed at the time when req_i will be placed at the end of the queue. For every A_{req_k} , we have that $A_{req_k} \leq C_{req_k}$. Let $Q' = [B_i | i = 0, \dots, M]$ be a sequence whose $B_0 = 0$. The remaining elements of Q' result from Q by removing the duplicate values existing in Q and sorting the remaining values in increasing order. Specifically, for Q' the following hold:

- (1) $B_0 = 0$
- (2) $\forall B_i, B_j \in Q' | i < j \Rightarrow B_i < B_j$
- (3) $\forall B_i > B_0$ there exists at least one $A_{req_k} \in Q | A_{req_k} = B_i$
- (4) $\forall A_{req_k} \in Q | (\exists B_i \in Q' | B_i = A_{req_k})$

For all $B_i \in Q'$ we define their multiplicity $mult_i$ as follows:

$$\begin{cases} mult_i = |Q_{B_i}|, i > 0 \\ mult_0 = 0, \quad i = 0 \end{cases}$$

where $Q_{B_i} \subseteq Q \wedge (\forall A_{req_k} \in Q_{B_i} | A_{req_k} = B_i)$.

Let $B_l \in Q'$ be the specific value that corresponds to the time units required for completing req_i , i.e., $B_l = A_{req_i}$, then the first B_1 units of B_l will actually execute in: $|Q| * B_1$ time units. After $|Q| * B_1$ time units, all the requests that required B_1 units to complete will be removed from the queue. Hence, the length of the scheduler's queue will become $|Q| - mult_1$. Moreover, all the requests that required B_2 time units to complete will now require $B_2 - B_1$ units. Consequently, the next $B_2 - B_1$ units of B_l will actually execute in $(|Q| - mult_1) * (B_2 - B_1)$. By induction, we can conclude that the overall service time D_{req_i} can be calculated using the following formula:

$$D_{req_i} = i + \sum_{k=1, \dots, l} ((|Q| - \sum_{m=0, \dots, l-1} mult_m) * (B_k - B_{k-1}))$$

In Figure 4, we revisit our case study scenario. Assume the situation discussed in Section 4.1 where the scientist wants to select out of three firemen the one that can fulfill his request, req , within the scientist's lifetime. This time the lifetimes of the three firemen are 9, 9 and 16 time units, respectively. The worst case execution time for req is 2 and the worst-case communication delay is 1. Suppose that the client assessor chooses Fireman 1 first. The scheduler of the

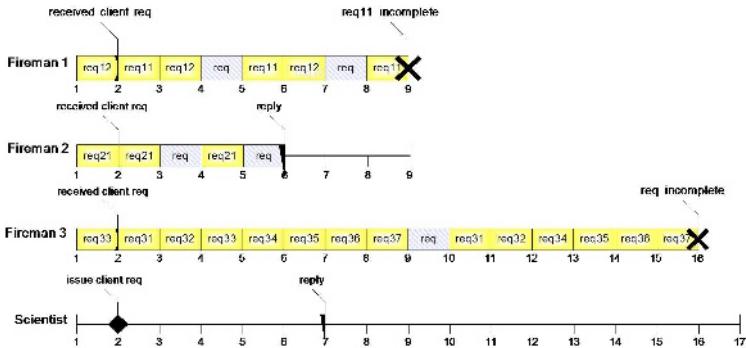


Fig. 4. Applying the LifetimeLoad policy

fireman has two pending requests in his queue. At the time when *req* arrives, *req₁₁* requires 4 more time units to complete, while *req₁₂* requires 2. With the addition of *req* in the queue, the overall delay for completing it shall be 6. Given that *req* arrives at time = 2, its execution will complete at time = 8. Consequently, the client will receive a reply at 9, which is legal. However, with the addition of *req*, the overall delay for completing *req₁₁* shall be 8 time units. Hence, *req₁₁* will finish at time 10, which is greater than the lifetime of the first fireman. Accepting, thus, *req* causes a missing deadline for the first fireman. Let us assume instead that the client assessor selects Fireman 2 first. The queue of his scheduler contains only one pending request that requires 2 more time units to complete at the arrival of *req*. The overall delays for completing *req* and *req₂₁* are 4 and 3, respectively. These values are legal with respect to the lifetimes of the client and Fireman 2. Consequently, *req* can be accepted.

4.3 EDFTB Policy

As we already discussed the LifetimeLoad policy provides quite strong guarantees. However, it can only provide them in cases of requests that are served once during the lifetime of a mobile server. In practice it is often the case that a mobile client requests a mobile server to perform a particular task more than once, usually with a certain period. In our example, for instance, the scientists may request the firemen to measure the level of radioactivity periodically and produce a certain amount of measures, which should be returned back to them towards performing more accurate analysis and estimations. Such kind of requests, leading to the execution of periodic activities, can not be guaranteed by the LifetimeLoad policy.

The EDFTB policy detailed here is suitable for both requests leading to periodic activities and typical requests that are served once. Hereafter, we call the former *periodic requests* and the latter *aperiodic requests*. As implied by the name of the policy, it relies on the classical EDF (Earliest Deadline First) [1] and the TB [7](Total Bandwidth) algorithms, which are customized here for the specific purpose of critical pervasive computing environments.

Briefly, Liu and Layland [1] examine a typical real-time system that executes only periodic tasks, which arrive dynamically in a queue. The execution of the tasks is preemptive. Every task t_i is characterized by a period T_i . Every instance of t_i must complete within T_i . Hence, T_i is the deadline for the completion of t_i . Moreover, t_i is characterized by a worst case execution time C_{t_i} . According to EDF, a task is scheduled if it is the one with the earliest deadline amongst all the periodic tasks in the queue. Liu and Layland proved that a particular set of tasks $\{t_1, t_2, \dots, t_N\}$ is scheduleable if and only if the system's utilization is at most 1. Formally:

$$U_P = \sum_{i=1, \dots, N} (C_i / T_i) \leq 1$$

In the TB algorithm, Spuri *et al.* [7] further consider a real-time system with both periodic and aperiodic tasks. To deal with this combination they propose dividing the system utilization into U_P , used for executing periodic tasks, and U_S , used for the execution of aperiodic tasks. Aperiodic tasks ta_i are given a deadline d_{ta_i} , on the basis of U_S . The given deadline is the shortest possible and does not jeopardize the execution of periodic tasks. Specifically:

$$d_{ta_i} = \max(r_k, d_{ta_{i-1}}) + C_{ta_i} / U_S$$

In the above formula, r_k denotes the time instant that ta_i arrived and $d_{ta_{i-1}}$ denotes the deadline given to the aperiodic task whose arrival immediately preceded the arrival of ta_i . Based on its given deadline, ta_i is scheduled by EDF as any periodic task. Given the previous, an overall set of periodic and aperiodic tasks is scheduleable if and only if $U_P + U_S \leq 1$.

In the rest of this section, we describe in detail our specific extension to the EDF and the TB algorithms to handle the case of critical pervasive computing environments.

With the EDFTB policy, the scheduler of mobile entities uses the EDF algorithm. However, the scheduleability of periodic and aperiodic requests further involves additional constraints, which are verified by the scheduler service upon the arrival of a new request req coming from the assessor of a mobile client. In particular, if req is periodic it will result in the execution of a periodic activity on the side of the mobile server. Since the lifetimes of both the client and the server are limited, the client is obliged to associate req with a period T_{req} and a required number of instances n_{req} of the periodic activity that is going to be executed. For example, a scientist should request a fireman to measure the level of radioactivity 3 times with a period of 2 time units. The request req is eventually received by the scheduler of m_{server} , which further assumes that the overall server utilization is divided into U_P and U_S for the execution of periodic and aperiodic requests, respectively. Given the previous, the goal of the scheduler is to verify whether the server can preserve the following constraints:

- For periodic requests:

1. req will be served n_{req} times within the lifetime of the server.
2. The replies to req will be send back to the client within the client's lifetime.

- For aperiodic requests:
 1. req will be served within the lifetime of the server.
 2. The reply to req will be send back to the client within the client's lifetime.

Based on the outcome of the verification procedure the scheduler sends a positive or a negative report to the client assessor. Depending on the report, the assessor behaves as in the case of the LifetimeLoad policy.

Periodic requests: In case req is periodic, the scheduler of m_{server} performs the following steps:

First it checks whether the utilization of m_{server} remains lower than 1 if the new request is accepted for service. Formally, if there exist $N_{m_{server}}$ pending periodic requests on the server, the scheduler verifies the following:

$$U_P = \sum_{i=1, \dots, N_{m_{server}}} (C_{req_i}/T_{req_i}) + (C_{req}/T_{req}) \leq 1 - U_S$$

In the above formula, C_{req_i} and T_{req_i} denote the worst case execution time and the period of each pending request. If this formula holds it means that the execution of req shall not jeopardize the execution of the rest of periodic requests that already exist in the scheduler's queue. However, the scheduler must further verify that the server's lifetime is sufficient to allow executing req n_{req} times. This is accomplished in the second step by evaluating the following:

$$m_{server}.lifetime/T_{req} \geq n_{req}$$

Hence, in the first two steps the scheduler of m_{server} verifies whether the server can guarantee the first of the two constraints stated for periodic requests. The third step performed by the scheduler amounts in checking the second constraint. Given that the execution of the activities that serve periodic requests is preemptive, the only way to assure that the client will get the replies to req within $m_{client}.lifetime$ is by checking whether m_{client} lives longer than m_{server} . More precisely, if c_{rep} denotes the worst case communication delay for sending a reply to req , then the following must hold:

$$m_{server}.lifetime \leq m_{client}.lifetime - c_{rep}$$

If all the above hold, the server reports back to the client with a positive answer. In the opposite case, the answer is negative and the client selects another candidate mobile entity. If the answers of all the alternative entities are negative the request is dropped by the client.

In our case study scenario assume the following situation, depicted in Figure 5. At time 2 a scientist wants to issue a periodic request req with $C_{req} = 1$ and $T_{req} = 4$ to one of the three firemen shown in the figure. The scientist further requires that req is executed 3 times. Suppose that Fireman 1 is selected first by the scientist. Fireman 1 periodically executes two requests req_{11} and req_{12} with $C_{req_{11}} = 2$, $T_{req_{11}} = 4$ and $C_{req_{12}} = 2$, $T_{req_{12}} = 4$, respectively. With the inclusion of req in the queue of Fireman 1, the U_P utilization shall become 1.5. Hence, Fireman 1 can not execute req without jeopardizing the execution of req_{11} and req_{12} . Assume instead that Fireman 2 is selected first by

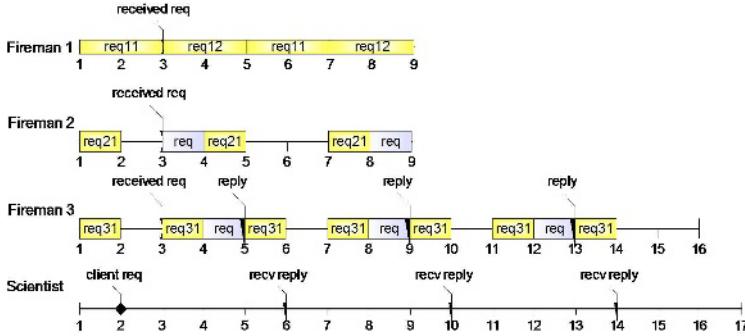


Fig. 5. Applying the EDFTB policy for periodic requests

the scientist. Fireman 2 is already responsible for the execution of one periodic request, req_{21} , with $C_{req_{21}} = 1$ and $T_{req_{21}} = 3$. Hence, with the inclusion of req the U_P utilization for Fireman 2 shall become 0.59. However, the remaining lifetime of the fireman at the time when he receives req is 6. Consequently, he can execute req at most 2 times. Fireman 3 in our example periodically executes one request with $C_{req_{31}} = 1$ and $T_{req_{31}} = 2$. With req , its utilization U_P , shall become 0.75. Moreover, the remaining lifetime of the fireman when he receives req is 13. Thus he can execute req 3 times. Finally, the lifetime of Fireman 3 is less than the lifetime of the scientist. If the communication overhead c_{rep} is at most 1, the fireman will deliver the replies to the scientist, within the scientist's lifetime. Summarizing, if the third fireman is the first entity contacted by the assessor component of the scientist, the report will be positive and req will be successfully executed.

Aperiodic requests: If req is aperiodic, the scheduler follows the TB approach [7]. Based on U_S , the scheduler assigns a deadline to req and checks whether this deadline is consistent with respect to the lifetimes of the client and the server. Formally, the deadline is given according to the following formula:

$$d_{req} = \max(r, d_{req'}) + C_{req}/U_S$$

In the above, r is the moment when req arrived and $d_{req'}$ is the deadline given to the aperiodic request req' whose arrival immediately preceded the arrival of req . Moreover, the following must hold to guarantee that with the given deadline a reply to req will be delivered back to the client, within the client's lifetime:

$$d_{req} \leq \min(m_{server}.lifetime, (m_{client}.lifetime - c_{req}))$$

In Figure 6 we revisit the situation discussed in Figure 5. In particular, the periodic request previously issued by the scientist is scheduled in Fireman 3 (req_{32} in the figure). Suppose now that the scientist further issues an aperiodic request req to the same fireman. For Fireman 3 we have $U_P = 0.75$ and $U_S = 0.25$. The request arrives at time 5. Since it is the first aperiodic request, it is given a deadline that is equal to 9 (i.e., $5 + 1/0.25$). If the worst case communication

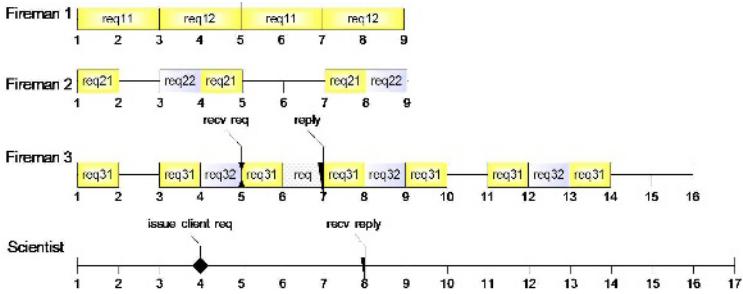


Fig. 6. Applying the EDFTB policy for aperiodic requests

overhead is 1, the scientist will receive a reply at time 10, at the latest. Consequently, *req* can be scheduled on Fireman 3 and the report sent to the scientist is positive. Actually, we can observe in the figure that *req* can be served earlier than 9 and the scientist may get the reply at time = 8.

4.4 Assessment

In Figure 7 we summarize our first experimental results produced by the application of the proposed policies in a simulated environment realized using the PARASOL simulator [8]. The environment consists of 3 mobile server entities, providing compatible services. We performed 2 different classes of experiments. In the first one (Figure 7(a)), we compare the performance of the Lifetime and the LifetimeLoad policies with 4 different workloads of *aperiodic* requests (100, 200, 400 and 800 requests, respectively). Similarly, in the second class of experiments (Figure 7(b)) we compare the performance of the Lifetime and the EDFTB policies with 4 different workloads of *periodic* requests (100, 200, 400 and 800 requests, respectively). The lifetimes of the mobile entities, the worst-case execution times for the requests and the periods of the periodic requests are randomly generated by following a uniform distribution. Our performance metric is *the percentage of accepted requests (i.e., the requests that are actually stored in the queue of a scheduler) that complete on time*, with respect to the lifetimes of the clients and the mobile servers.

Specifically, in Figure 7(a) we observe that the percentage of aperiodic requests that complete on time in the case of the Lifetime policy, linearly decreases as we increase the number of requests that constitute the overall workload issued to the mobile servers. As opposed to that, the LifetimeLoad policy guarantees that all the accepted requests are executed on time (i.e., we do not have any missed deadlines). In Figure 7(b) we observe that the behavior of the Lifetime policy gets even worst, given that the requests are periodic and require more time to complete. The EDFTB policy behaves perfectly in this case as we observe that all the accepted requests finish on time. The price to pay for avoiding missed deadlines is given in Figure 7(c). As mentioned in Section 1, we can estimate the battery overhead introduced by the three policies by examining the number of messages exchanged between a mobile client and a mobile server. In the

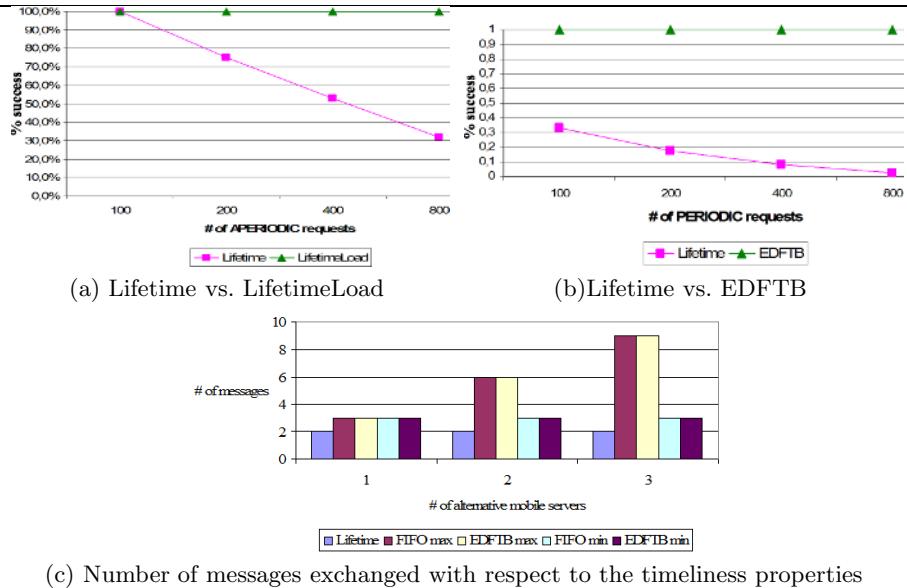


Fig. 7. Experimental results

figure we can observe that the overall number of messages exchanged between a mobile client and the alternative mobile servers is constant in the Lifetime policy, which is the cheapest. For the other two policies the minimum number of messages exchanged is greater but is still constant. On the other hand, the maximum number of messages for the LifetimeLoad and the EDFTB policies linearly increases with the number of alternative mobile servers.

5 Related Work

Up to know, there have been several approaches dealing with various dependability attributes in the context of pervasive computing environments [9,10]. These approaches most frequently concentrate on reliability, performance, availability, security, reputation, etc. However, to our knowledge the approach proposed in this paper is the first attempt that focuses on timeliness.

In the past, the middleware community proposed standards for real-time middleware platforms, which were aimed at conventional distributing computing environments where both the client and the server entities are deployed on top of stationary workstations. Among these standard approaches we have the one proposed by the OMG for Real-Time CORBA and related implementations like TAO and ZEN [11,12], which proved to be useful in conventional systems. However, they can not be directly employed in the environments examined in this paper.

In the remainder of this section we concentrate on work that is complementary to the approach we propose. In particular, the issue of calculating the

lifetime of mobile entities is central to our approach. Currently, we have useful techniques that aim at estimating the physical motion of mobile entities [3] and motion independent techniques that estimate the unavailability of mobile entities [13]. Several classical algorithms for scheduling real-time tasks have been proposed in the past. Usually they are divided into static and dynamic. Obviously, static algorithms can not be used in critical pervasive computing environments. Dynamic scheduling algorithms, schedule tasks on-the-fly. The EDF algorithm that we employed in this paper is among the most widely known ones. However, several others like MLF (minimum-laxity-first) and MUF (maximum-urgency-first) [14,15] may prove useful in the context of critical pervasive computing environments. As in the case of EDF, these algorithms should also be appropriately enhanced before they are introduced in such environments. In our particular case, we consider using MUF instead of EDF to deal with cases where all of the alternative mobile servers are incapable of serving a new client request. MUF associates tasks with an importance factor, which may serve as a criterion for rejecting pending requests towards serving new ones of greater importance.

6 Conclusion

In this paper, we proposed a middleware service that facilitates the timely execution of mobile services in critical pervasive computing environments. The overall service architecture relies on the WSAMI platform and supports three different timeliness policies for the execution of requests issued by mobile clients to mobile servers. The first policy takes into account the lifetimes of the client and the server entities and guarantees that a reply will be sent to the client as long as the server manages to serve the client's request. The second policy guarantees both that the client request will be served and that a reply will be sent back within the client's lifetime. This is achieved by examining the servers' load along with the client and the servers lifetimes. The third policy extends the classical EDF and TB algorithms for the purpose of pervasive computing environments, to deal with periodic requests in such environments. So far, the proposed service deals with the timely execution of independent client requests. An interesting extension would be to further incorporate support for the timely execution of workflows [16].

Acknowledgments. This work is partially funded by the MobWS GSRT grant for Cooperation in S&T areas with European countries.

References

1. C. L. Liu and J. W. Layland: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM* **20** (1973) 46–61
2. E. Pitoura and G. Samaras: Data Management for Mobile Computing. Kluwer Academic Publishers (1998)
3. H. M. O. Mokhtar and J. Su: Universal Trajectory Queries for Moving Object Databases. In: Proceedings of the IEEE International Conference on Mobile Data Management (MDM'04). (2004) 133–146

4. V. Issarny, D. Sacchetti, F. Tartanoglu, F. Sailhan, R. Chibout, N. Levy and A. Talamona: Developing Ambient Intelligence Systems:A Solution Based on Web Services. *Journal of Automated Software Engineering* **12** (2005) 101–137
5. W3C: Web Services Architecture. Technical report, (W3C) <http://www.w3.org/TR/ws-arch/>.
6. IEEE: IEEE Standard for Wireless LAN Medium Access Control (MAC). Technical report, IEEE (1997)
7. M. Spuri, G. Buttazzo and F. Sensini: Robust Aperiodic Scheduling under Dynamic Priority Systems. In: Proceedings of the 16th IEEE Real Time Systems Symposium (RTSS'95). (1995) 210–221
8. J. Neilson: PARASOL Users' Manual (v 3.1.). Technical report, (School of Computer Science - Carleton University - Ottawa) K1S5B6.
9. J. Liu and V. Issarny: QoS-Aware Service Location in Mobile Ad-Hoc Networks. In: Proceedings of the 5th IEEE International Conference on Mobile Data Management (MDM'04). (2003)
10. L. Zeng, B. Benatallah and M. Dumas: Quality Driven Web Services Composition. In: Proceedings of the 12th ACM International Conference on the World Wide Web (WWW'03). (2003) 411–421
11. R. E. Schantz, J. P. Loyall, D. C. Schmidt, C. Rodrigues, Y. Krishnamurthy, and I. Pyarali: Flexible and Adaptive QoS Control for Distributed Real-time and Embedded Middleware. In: Proceedings of the 4th IFIP/ACM/USENIX International Conference on Distributed Systems Platforms (Middleware'03). (2003)
12. A. Krishna, D. C. Schmidt, and R. Klefstad: Enhancing Real-Time CORBA via Real-Time Java. In: Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04). (2004)
13. Y. Xiong, X. Lin and J. Rowson: Estimating Device Availability in Pervasive Peer-to-Peer Environment. In: Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04). (2004)
14. M. L. Dertouzos and A. K. L. Mok: Multiprocessor On-Line Scheduling of Hard Real-Time Tasks. *IEEE Transactions on Software Engineering* **15** (1989) 1497–1506
15. D. B. Stewart and P. K. Khosla: Real-Time Scheduling of Sensor-Based Control Systems. In: Proceedings of the 8th IEEE International Workshop on Real-Time Operating Systems and Software (RTOSS'91). (1991)
16. A. Zarras, P. Vassiliadis and V. Issarny: Model-Driven Dependability Analysis of Web Services. In: Proceedings of the 6th International Conference on Distributed Objects and Applications (DOA'04). (2004) 1608–1625

Mobility Management and Communication Support for Nomadic Applications

Marcello Cinque¹, Domenico Cotroneo¹, and Stefano Russo^{1,2}

¹ Dipartimento di Informatica e Sistemistica,
Università degli Studi di Napoli Federico II,
Via Claudio 21, 80125 - Naples, Italy

{macinque, cotroneo, sterusso}@unina.it
² Laboratorio ITEM,

Consorzio Interuniversitario Nazionale per l'Informatica,
Via Diocleziano 328 - 80124 Naples, Italy

Abstract. There is an increasing demand for realizing communication services for nomadic environments, capable to provide applications with mobility management facilities and application-aware adaptation support. This paper proposes a novel mobility management and communication architecture specifically suited for nomadic environments, offering communication facilities and adaptation support by means of an API, named NCSOCKS. The driving idea is to provide application and middleware developers of nomadic services with essential mobility-enabled communication support, while hiding network heterogeneity in terms of wireless technology and leveraging the availability level of communication in spite of transient signal degradations. Transient signal degradations, due to device movements and/or shadowing, have the effect of increasing the handoff frequency. The proposed architecture integrates a novel mechanism to improve the connection availability by reducing the number of unnecessary handover procedures. In order to evaluate the proposal, an approach based on combined use of simulation and prototype-based measurements is adopted.

1 Rationale and Contributions

Recent years have been characterized by an explosive growth in the deployment of wireless data communication technologies. The IEEE 802.11 family [1], Bluetooth [2], and the infrared wireless technologies [3] are nowadays widely deployed, and backed by a growing number of hardware and software vendors. This scenario has led to the definition of a new computing paradigm, the Nomadic Computing.

Nomadic Computing (NC) is a form of mobile computing where communication takes place over strongly heterogeneous network infrastructure, composed of several wireless domains or cells, which are glued together by a permanent network infrastructure (the core network), aiming to provide anytime, anywhere access to mobile devices [4]. A typical NC infrastructure model is illustrated in

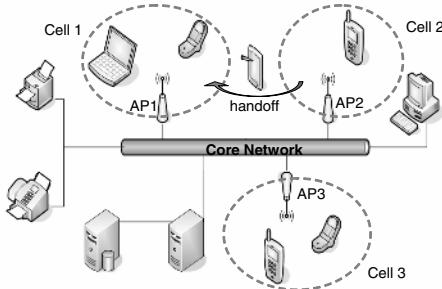


Fig. 1. Nomadic Computing Model

figure 1. As figure shows, devices can move among different cells, with different technical characteristics. Each cell is covered by one or more Access Points (APs), such as Bluetooth, Wi-Fi, and IrDA, each one providing devices with the access to services deployed over the core network. The wireless network heterogeneity of nomadic environments is coherent with the Next-Generation Wireless Internet (NGWI) view of mobility [5] which aims to provide Internet access through a diverse set of wireless architectures or “meshed networks”, e.g., wireless local area network (Wi-Fi, Bluetooth), 3G cellular, and satellite network.

Nomadic Computing is enabling the creation of a new generation of applications; examples are applications for guiding tourists as they move among halls, or locations, of a touristic site (e.g. a museum or an archaeological site), applications for context-aware medicine containers, or bed in a hospital [6]. Other scenarios are network-based dependable control [7], and aircraft maintenance systems [8]. In such scenarios, providing high available communication services is a crucial requirement. Furthermore, the network heterogeneity, along with devices’ ability to move among different cells, increases the need to keep applications aware of the current state of the connection and device’s location.

Several research studies have been conducted on various aspects of nomadic environments. Recently, research efforts have progressed along mobility management issues. Mobility management can be decomposed in handoff management, and location management. Handoff management’s objective is to support users with mobile terminals to access services through wireless networks when they move to a new service area. Most research studies focused on efficient mechanisms provision, independence of the network technologies, to support both horizontal and vertical handoffs [9, 10]. Other studies moved toward the definition of communication protocols, capable of leveraging quality attributes, such as availability, reliability, and transparency [11, 12, 13]. Location management is concerned with the provision of the physical or symbolic location of mobile terminals. There are studies proposing new strategies or mechanisms, and most of them assume the use of special or intelligent sensors, as described in [14], where a thorough analysis of different location mechanism has been made. Other proposals use Commercial Off The Shelf (COTS) sensors [15, 16], thus they do not

mandate the use of a dedicated wireless infrastructure for locationing purposes. By location support, we hereafter refer to the symbolic location support, that encompasses abstract ideas on where something is located [14]: in the kitchen, in the lab, next to a picture in a museum.

These research studies represent fundamental milestones for achieving new solutions and clarifying the open issues to be addressed. Nevertheless, we recognize the lack of mobility management and communication support able to face the following fundamental NC-related issues in an integrated manner: i) provide applications with adaptation support to the highly variable network conditions; ii) leverage communication availability in spite of transient signal degradations and AP failures; and iii) provide a technology-independent Application Programming Interface (API).

This paper presents the design and the implementation of a mobility management and communication architecture specifically tailored for nomadic environments. Design choices that have been made to cope with the above mentioned issues are discussed. The paper describes the proposed technology-independent architecture along with its object-oriented communication API, called Nomadic Computing Sockets (NCSOCKS¹), which provides applications with adaptation mechanisms to connection status and location changes (connection and location awareness support). In order to achieve availability of communication, the architecture integrates a new mechanism to reduce the connection unavailability due to unnecessary handoff procedures, i.e. handoff procedures triggered by transient signal degradations. Experimental results to evaluate availability improvement and performance penalty are provided over a testbed composed of Bluetooth and Wi-Fi appliances.

The proposed architecture can be thought as a building block for the implementation of middleware solutions for nomadic environments. Furthermore, design choices can be used as a guideline for software practitioners to solve similar problems, thus reducing the development efforts.

2 Mobility Management and Communication Support

2.1 Assumptions and Requirements

Before illustrating in detail the proposed mobility management support, we need to clarify assumptions we make about the supporting infrastructure and requirements we needed to satisfy. As far as network technology is concerned, we assume the presence of heterogeneous wireless networks, consisting of short range COTS wireless access point. In other words, we do not mandate the adoption of a particular access point equipped with intelligent or special sensors. Conversely, we assume that wireless networks are equipped with COTS (and cheap) access points. As far as the core network is concerned, we assume that it is wired

¹ The NCSOCKS API along with its documentation can be downloaded from <http://www.mobilab.unina.it/Prototypes.htm>

and reliable. As for location management, throughout this paper the focus is on indoor symbolic location management.

The following requirements have been considered designing the solution:

- Provide applications with adaptation support to the highly variable network conditions in terms of connection status (availability, signal strength, bandwidth, delay) and device's location. The connection and location awareness support allows applications to promptly adapt to the varying conditions, e.g. the guide application can stream to the device the audio-video guide for the current room, while adapting the frame rate to the current delay.
- Leverage communication availability in spite of transient signal degradations, due to device movements and/or shadowing, and of crash of network access points. This represents a fundamental requirement for mission critical nomadic applications.
- Provide a communication Application Programming Interface (API) which is independent from the heterogeneous wireless network infrastructure. The need of creating a technology-independent software API for NC environments is strongly recognized. As stated in [17], such an API *will encourage the creation of long lived applications and of new location-aware application by helping to amortize development efforts*.

2.2 Overall Architecture

The proposed mobility management and communication architecture is depicted in figure 2. The architecture is composed of a mobile-side platform and core-side components. The mobile-side platform offers services to nomadic applications, whereas the core-side components support mobile-side components to perform their tasks.

With regard to the mobile-side platform, we adopted a cross-layer design approach. The NCSOCKS transport layer offers services to the applications by using both the network layer and the data-link layer. The interaction with the data-link layer is made necessary to gather information about the current state of the connection and device's location, enabling connection and location awareness. The NCSOCKS collects data-link related information via the Connection

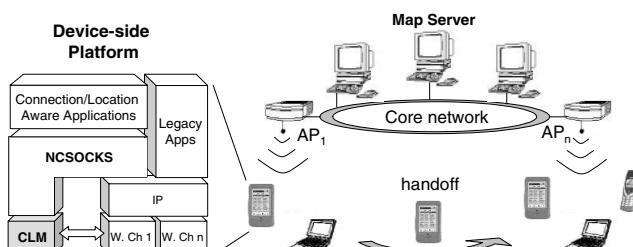


Fig. 2. Overall Mobility Management Architecture

and Location Manager (CLM), which manages the multiple, heterogeneous wireless interfaces of the device, and performs handoff operations among different APs during device's movements. This way, the above layers do not have to be aware of the particular wireless interface being used, overcoming heterogeneity issues. The CLM is also aware of the current location of the device, in terms of the AP that the device is using, and it provides the NCSOCKS with such location information. As for the network layer, the Internet Protocol (IP) is adopted, for several reasons. First, using IP masks the heterogeneity of the underlying wireless technology. Second, it allows the plenty of legacy IP-based applications to run on mobile terminals while they move, coherently with the NGWI view. The handoff is managed by the CLM and it is thus completely transparent to legacy applications. On the other hand, legacy applications cannot benefit from the connection and location awareness support, and will suffer the high variability of network conditions. Third, using IP does not represent a limitation, since it is supported by the majority of COTS wireless technologies, including Bluetooth (IP over L2CAP via BNEP encapsulation [2]), Wi-Fi (via a native support), and IrDa (IP over serial communication [3]).

As will be clarified in section 3, the CLM uses information about the topology of APs to perform the handoff. For this reason, components are deployed on the core network in order to provide the CLM with the topology map of the environment. In particular, the Map Server provides information about the current topology, including the description of the communication technology being used by each AP, and it keeps the map updated in spite of AP unavailabilities due to AP failures or overload conditions. Map Server's services can also be used to implement administrative tools to monitor the status of the network in terms of AP activity and the devices connected to the wireless network. Due to scalability purposes and/or topology constraints, the nomadic computing infrastructure can be scattered into two or more wireless service areas, each one served by its own Map Server.

2.3 CLM

As described earlier, the CLM component is in charge of implementing handoff procedures. These procedures are composed of the following three phases [18]:

- Initiation: the network status is monitored to decide for a migration;
- Decision: once the need for handoff is triggered, a new AP has to be selected;
- Execution: the connection to the selected AP is established and current location is updated.

The initiation phase is strongly dependent on the technology being used. For instance, in the case of a Bluetooth cell, in which the slave mobile device can manage at most one connection per time, the initiation phase can use only the information about the current connection. In the decision phase, a new AP is selected. The decision can be taken by monitoring a set parameters of the wireless link between mobile device and APs (see section 3.3). In order to reduce

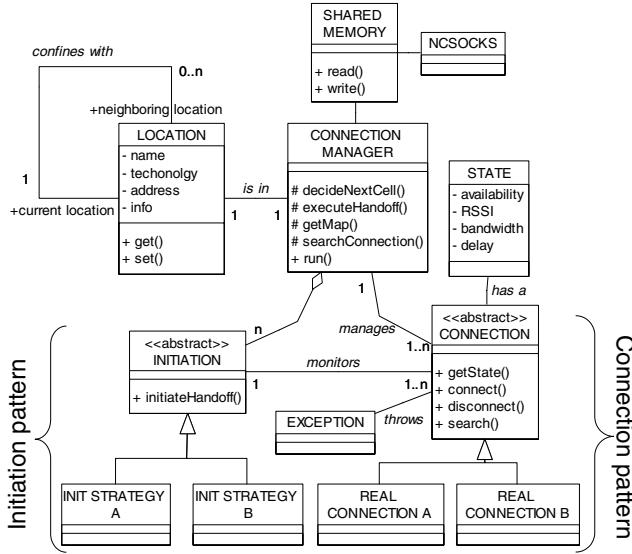


Fig. 3. CLM Class Diagram

the number of APs to be inquired, a predictive handoff scheme is adopted. This scheme uses services provided by the Map Server to obtain information about neighboring APs.

We herein present the design of the CLM layer as a design pattern. This enable us to propose a general solution which can be adopted by designers facing similar problems. The class diagram of CLM is depicted in figure 3. The **Connection Manager** holds information about the connection status, current location, and technology being used. The connection status is composed by several data: the availability, the signal strength, the bandwidth, and the delay. The availability can assume three distinct values: i) *NOT CONNECTED*, i.e. the device is not connected to any AP; ii) *HANDOFF*, i.e. the mobile device is performing a handoff procedure; and iii) *CONNECTED*, i.e. the device is connected to an AP. Two strategy patterns are adopted, called *Initiation* and *Connection*. The former aims to define a family of initiation algorithms, encapsulate each one, and make them interchangeable in spite of the technology being used. The latter provides the wireless connection abstraction, in terms of connection creation, disconnections, AP discovering, and monitoring procedures for signal strength, bandwidth, and delay. Technology specific wireless APIs are encapsulated in **Real Connection** objects. Mismatches may occur between the functionalities provided by the vendor-specific API and the abstract **Connection** class methods. For instance, a technology may offer additional facilities. However, they can be ignored since they are not needed by the **Connection Manager** class. More complex is the case in which a technology offers only a limited support as related to the **Connection** class. For example, a technology may not provide

any API to obtain the signal strength. In this case one can either i) perform a work-around (e.g. reading and parsing the output of a command line tool which provides the signal strength), or ii) avoid to implement the functionality. In this last case, applications should be notified about the absence of the signal strength information for that particular technology, and also the **Connection Manager** should be tailored with respect to the missing information. The topology map is implemented by means of **Location** objects and built by invoking the `getMap()` method, which has the effect of requesting the overall map to the Map Server serving the current wireless service area.

As for the CLM's dynamic behavior, the Connection Manager evolves according to the following steps: i) search for an AP and topology map retrieval (via the `getMap()` method); ii) handoff initiation; and iii) handoff decision and execution: if this last step succeeds, the manager jumps back to the step ii), otherwise it starts again from the step i). Therefore, the step i) is performed either when the mobile device enters the wireless infrastructure for the first time or when the handoff procedure fails. A failure during the handoff process may be due to an AP failure (which requires the map information to be updated), or may due to the mobile device going beyond the zone covered by the current wireless service area. In this last case, it is worth to start from step i), since the device may be moving towards another wireless service area.

The CLM pattern is characterized by: i) *Generality*, i.e. the structure and behavior of the **Connection Manager** and **Location** are independent of the technological details, and abstracted by the **Initiation** and **Connection** classes, and ii) *Extensibility*, i.e. it is possible to add new **Initiation** and **Connection** classes in charge of managing new wireless communication technologies.

2.4 NCSOCKS

The class diagram of the Nomadic Computing Sockets (NCSOCKS) is depicted in figure 4. As figure shows, the API is similar to the standard Java sockets library, except for the following.

First, the NCSOCKS have to provide connection and location awareness. This is done through two classes, the **Location Monitor** and the **Connection Monitor**. Applications can request the current connection status or location either via synchronous or asynchronous primitives. For example, an application can request the location by simply invoking the `getActualLocation()` method or, alternatively, it can subscribe a handler (implementing a **Callback** interface) in order to be notified about location changes by means of the `notifyLocChanges()` method. The information about the current location and connection status are gathered by the NCSOCKS from the CLM by means of a shared memory abstraction. The **Shared Memory** class encapsulates the mechanisms to deal with the NCSOCKS and CLM reader/writer concurrency problem.

Second, communication methods have to take into account the variability of the connection status in terms of availability adopting proper synchronization paradigms. We adopted the following synchronization paradigm. The `Send()` primitive tries to send a packet (a segment, in the case of TCP) even if the

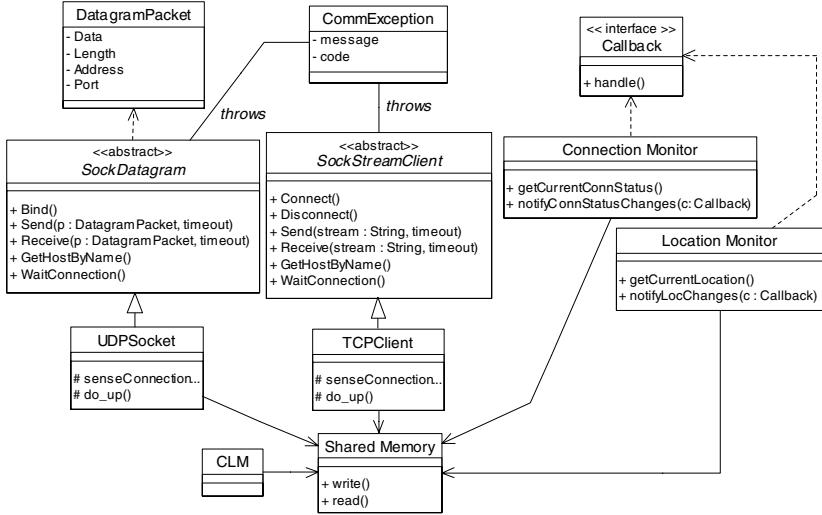


Fig. 4. NCSOCKS Class Diagram

actual state is HANOFF or NOT CONNECTED, in the sense that it waits (by blocking the caller) for a time slot, storing the packet (the segment) into a buffer. Once the connection reaches the CONNECTED state, the stored packet (segment) is effectively sent. Conversely, when a timeout expires, the application is notified via an exception with the current connection status, and the packet (segment) is dropped. From this point on, applications can adapt their behavior accordingly. For instance, a multimedia streaming application may decide to not re-transmit the lost packets, because the delay due to a re-transmission might be not acceptable for its quality requirements. Conversely, content-type applications, such as smart guides, may decide to re-transmit, in order to send all the contents to the user. The `Receive()` primitive works differently: it receives packets only when the state is CONNECTED, otherwise it notifies the application via an exception that the connection is not available anymore. This choice is due to the following consideration: if the connection is unavailable, and the `Receive()` does not notify the application, the application will waste resources waiting for incoming packets. Instead, during unavailability periods, the application can perform other tasks (e.g. backup the current status, or start to elaborate the already received data). In the worst case in which the application does not have any tasks to perform, it can wait for the connection to become available through the `WaitConnection()` method.

2.5 Map Server

The Map Server is a core-side component, which is in charge of providing the mobile devices, i.e. the `Connection Manager` object, with the current environmental topology map. Each mobile terminal is identified by an unique identifier.

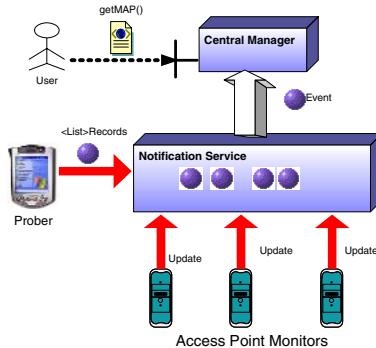


Fig. 5. Overall Architecture of the Map Server

The Map Server is designed as a set of distributed objects: one object per AP (**AP monitors**), which are in charge of performing the AP monitoring tasks, and one **Central Manager** which is in charge of creating and keeping up to date the topology map. This can be managed by the system administrator via the **Central Manager** user interface, whereas the **AP monitors** notify the manager about current APs status (see figure 5). More precisely, AP monitors are in charge of notifying the **Central Manager** about topology changes due to AP crashes or overloads (i.e. the maximum expected number of devices is connected to the AP). Thus, topology changes not handled by human administrators are automatically handled by the **Central Manager** by means of AP monitors. The notification takes place via an Event Channel.

The map discovery process takes place via a prober application, as illustrated in figure 6. It is an application which resides in a mobile terminal, which is in charge of collecting measurements with the human support. Measures consist of normalized Receiver Signal Strength Indicator (RSSI) values as received by the prober. Such measures are gathered by the prober along the path enforced by the human operator and, at the end of the discovery phase, they are sent to Central Manager via the Event Channel in order to create the topology map. For each room of the nomadic environment, the prober discovers its APs (selecting them among the others on the basis of the RSSI) and, for each neighboring zone between two rooms, neighboring APs are discovered, building the map. The measurements gathered in the neighboring zones are useful to tune the location system precision, as will be explained in section 5.3.

The approach is general enough to deal with different AP topologies. We assume that each room can have one or more APs, which together build a cluster of APs for the room. This way, overlapping cells are taken into account as clusters. The neighboring APs cluster for each cluster is discovered by the prober along the path enforced by the human operator. Therefore, the virtual map is coherent with the physical environment within which the mobile device roams, regardless to its shape and/or extension. Finally, if two covered zones are separated by an

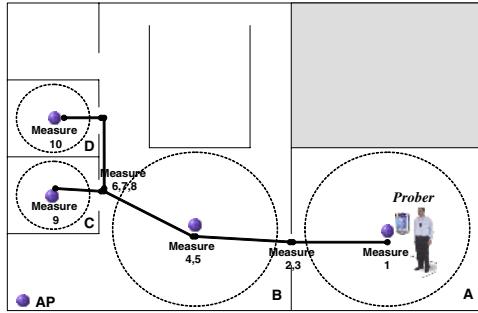


Fig. 6. The Topology Discovery process

uncovered area, the two zones can be seen as two different wireless service areas and can be served by two different Map Servers.

3 Handoff Strategy

This section describes an handoff strategy, implemented by the CLM. The presented strategy is designed with two objectives in mind: i) to improve the network availability in terms of wireless connection, and ii) to use of the closest AP, in order to keep track of device's location. Nevertheless, other handoff strategies pursuing different goals can be designed and integrated in the architecture.

3.1 Handoff Assumptions

Handoff procedures can be classified according to the level at which they operate [10, 19, 13]. We refer to an L_n handoff for a handoff procedure that works at level n of the ISO-OSI stack. Focus here is on L2 handoff, i.e. vertical (i.e., handoff performed between two distinct technologies) and horizontal (i.e., among the same technology) handoff performed at data link layer. As far as initiation strategies for vertical handoff are concerned, we assume that they are based on signal strength measurements in the cell being used. This results in lower power consumption (one interface at a time is adopted), and in interference reduction (for instance, using Bluetooth and Wi-Fi simultaneously may produce significant interference [20]). As for horizontal handoff, some technologies (e.g. Wi-Fi and cellular technologies, such as GSM) manage the handoff in a transparent manner, so we only have to keep track of the AP being connected. There also are other technologies (such as Bluetooth and IrDA) in which the handoff is not supported. In these cases, we handle handoff procedures in the same manner as the vertical ones: as an example, when a device in a Bluetooth cell triggers a handoff, it can reconnect either to a Bluetooth AP or to a different one.

In order to pursue the portability of legacy IP-based applications, handoff at network (L3), at transport (L4), or at session layer (L5) should be supported

as well. To this aim, the proposed L2 handoff can be integrated with existing solutions: i) L3 handoff protocol, such as Mobile IP [12], which is a general technique to perform handoff of IP traffic between IP subnets using straightforward routing techniques and IP-IP encapsulation; ii) L4 handoff, like TCP/DNS based handoff protocols, that is vertical handoff at the TCP level [10]; and iii) L5 handoff, such as SIP-based handoff procedure [13]. However, L2 solutions are enough to support UDP and stateless TCP applications with a minimal configuration effort at network level (i.e. using DHCP or configuring the IP address by means of Zero Conf IP - <http://files.zeroconf.org>).

3.2 Initiation Strategy

The connection availability is strongly dependent by the number of handoff operations. The more handoffs are performed, the less the connection is available. Defined $P_r(H)$ as the probability of initiating an handoff, improving the availability means minimizing the probability $P_r(H)$. Several proposals of initiation strategies, referred in literature, are characterized by assumptions similar to those we made, that is, the handoff recognition is performed by using signal information of the cell currently in use. Reactive approaches, based on broken link recognition, such as [10], initiate the handoff in spite of AP unavailability detection. Proactive approaches allow the handoff to be triggered when the AP is still available. For this reason, this approach should be preferred. In this direction, some solutions are based on a simple threshold mechanism, that is the handoff is initiated when the Receiver Signal Strength Indicator (RSSI) falls below a certain threshold [18], [21]. In this case, the value of $P_r(H)$ is calculated as:

$$P_r(H) = P_r(RSSI < S_{RSSI}) \quad (1)$$

where S_{RSSI} is a fixed threshold on RSSI. We can argue, as experimental results confirm (see section 5.2), that this kind of initiation leads to a poor availability. Indeed, noisy environments and shadowing problems can lead to transient RSSI degradations, which do not strictly require any handoff.

For this reason, it becomes crucial to define a mechanism which is able to discriminate permanent signal degradations from transient ones. Some strategies involve averaging the signal received with different window size, to ensure the accuracy of the handoff initiation procedure using a fuzzy logic controller [9]. Among the heuristics based on the concept of threshold, the α -count mechanism (already adopted in other research areas, such as intermittent failures detection [22]), appears to be particularly interesting for our purposes due to the clear and simple mathematical characterization, the thorough analysis already conducted, and the minimal computational complexity. We propose an α -count function which is RSSI-based, in order to keep the device connected to a close AP, for locationing purposes. The RSSI is strongly related to the distance between the device and the AP. The proposed α -count function is defined as follows:

$$\alpha^{(L)} = \begin{cases} \alpha^{(L-1)} + 1 & \text{if } RSSI^{(L)} < S_{RSSI} \\ \alpha^{(L-1)} - dec & \text{if } RSSI^{(L)} \geq S_{RSSI} \text{ and } \alpha^{(L-1)} - dec > 0 \\ 0 & \text{if } RSSI^{(L)} \geq S_{RSSI} \text{ and } \alpha^{(L-1)} - dec \leq 0 \end{cases}$$

During the L -th measurement, if RSSI falls below the threshold, the value of the $\alpha^{(L)}$ function is incremented, otherwise the value is decremented by a positive quantity dec . A handoff is triggered when $\alpha^{(L)}$ becomes greater than a threshold α_T . The α_T , dec , and S_{RSSI} values have to be carefully tuned in order to achieve a trade-off between the availability and accuracy of the location mechanism. Indeed, α_T along with values of dec and S_{RSSI} indicates which is the zone covered by an AP.

Transient degradation of RSSI causes $\alpha^{(L)}$ to be incremented, and successively decremented. The handoff probability can be calculated as follow:

$$\begin{aligned} P_r(H) &= P_r(RSSI^L < S_{RSSI}, \alpha^{(L-1)} \geq \alpha_T - 1) = \\ &= P_r(RSSI^L < S_{RSSI}) \cdot P_r(\alpha^{(L-1)} \geq \alpha_T - 1 | RSSI^L < S_{RSSI}) = \\ &= P_r(RSSI^L < S_{RSSI}) \cdot P_r(\alpha^{(L-1)} \geq \alpha_T - 1) \end{aligned} \quad (2)$$

This probability is equal to the $P_r(H)$ as evaluated in (1) in the case of simple threshold, multiplied by $P_r(\alpha^{(L-1)} \geq \alpha_T - 1) \leq 1$. As we expect, $P_r(H)$ is lower than the simple threshold strategy. Hence, the α -count results in a reduction of the number N_H of unnecessary handoff operations, as experimental results confirm (see section 5.2).

3.3 Decision Algorithm and Locationing Issues

The decision phase is based on a topology-based schema. The decision algorithm is in charge of electing the new AP among the neighboring APs. The decision is taken by using a score criteria: let $N = \{ng_1, \dots, ng_n\}$ be the set of neighboring APs, each one belonging to a certain cluster. For each $ng_i \in N$ a score $s(ng_i)$ is evaluated on the basis of several parameters (RSSI, delay, bandwidth). The decision algorithm selects the AP ng^* with the score $s^* = \max_{ng_i \in N} s(ng_i)$.

As for locationing issues, we assume that a mobile device is in room x when it is attached to a AP belonging to the cluster x . The initiation phase assures that when a mobile device leaves a room, a handoff will be triggered. The decision algorithm assures that when a mobile device enters in a room y , with APs belonging to the cluster y , one of those APs will be selected. The score parameters used by the decision algorithm are strongly influenced by the distance between the device and the AP, as well as by the presence of walls, as several research studies, such as [20], confirm. For this reason, being the APs of the cluster y closer than other APs belonging to other clusters, their score should be the best. However, even if pathological situations can lead to the selection of a wrong AP, poor values of the signal strength, which are measured on the selected AP, will eventually result in the initiation of another handoff, thus correcting the error.

It is worth noting that, since it is likely that each AP cluster has only a few neighbors (typically from 1 up to 4, such as a medium office environment), the decision time spent is minimized as compared with that needed for scanning all the APs in the environment, as some solutions suggest [9]. Furthermore, we point out that location management mechanism do not mandate the adoption of dedicated sensors, since they use the same sensors forming the communication

cell infrastructure. Further details about the location support can be found in our previous work [23]. Finally, due to the technology-independent characteristics of the CLM, this solution does not depend on the wireless technology being adopted, resolving the heterogeneity problems in location aware computing, often mentioned in literature, as shown in [14, 17, 16].

4 Implementation Issues

The CLM has been conceived as a daemon process running on mobile devices. Communication between user applications and the CLM has been implemented by using shared memory system calls, which are encapsulated in the `SharedMemory` class (see figure 3). The Bluetooth connection is achieved by a specific `PANConnection` class, which implements the `Connection` abstract class. Such a class is in charge of creating “IP over L2CAP” channels based on Bluetooth Personal Area Network profile (PAN) and Bluetooth Network Encapsulation Protocol (BNEP) [2]. For this purpose, we adopted the BlueZ API (the standard Bluetooth support for Linux - <http://bluez.sourceforge.net>) which allowed to implement all the methods required by the `Connection` class.

As far as Wi-Fi is concerned, the implementation of components (a `WiFiConnection` class implementing the `Connection` class) was simplified by the fact that the IP abstraction is natively provided by Wi-Fi adapters. Nevertheless, Wi-Fi does not provide any standard API to program the adapter. For this reason, we used command line tools (e.g. activating the interface to connect) directly from within the `WiFiConnection` class methods.

The α -count based initiation strategy explained in section 3.2 has been implemented through a `alphaInitiation` class which implements the `Initiation` interface. In the first release of the architecture, we initiate handoffs with the α -count initiation strategy irrespective from the technology being used. This may represent a problem when using wireless data link technologies which automatically manage the horizontal L2 handover (i.e. Wi-Fi). We define three strategies to overcome the problem: i) L2 technology could be forced to not perform L2 horizontal handoffs, which will be treated by the `Connection Manager` (the main weakness is performance losses); ii) it could be assumed that the neighboring cells of the considered technology forms an unique big cell (the main weakness is locationing accuracy losses); and iii) it could be forced a topology in which each AP has neighbors that are all of different technologies (the main weakness is to introduce hardware constrains in the communication infrastructure topology). The most suitable strategy can be chosen on the basis of the requirements or constraints forced by the system administrator.

5 Experimental Results

5.1 Testbed Description

A testbed was established in the laboratories of department building. The layout of these labs is shown in figure 6. It has dimensions of 30m by 40m with about 6

different rooms, including computer labs, offices, and the storeroom. As shown in the figure, APs are placed in the middle of four rooms. The APs used were three ANYCOM Bluetooth dongles (in rooms B, C, and D) and an Orinoco Access Point 802.11b (in room A). As for mobile device's platform, we used a Compaq iPAQ 3970 mobile device equipped with a Bluetooth and 802.11 modules, and with the Linux Familiar 0.7.0 operating system.

5.2 Availability and Performance Measurements

The proposed α -count scheme is compared with the fixed threshold mechanism (that is, an handoff is triggered when the RSSI value is lower than a fixed threshold), in order to evaluate the availability improvement. In particular we show unavailability reduction by evaluating the $P_r(H)$ (i.e. the probability that an handoff occurs). The initiation strategies are compared as a function of S_{RSSI} , since this parameter is the only one that affects the fixed threshold scheme. A simulation model is developed using the MATLAB environment (<http://www.mathworks.com>). Such a model performs the following two tasks: i) simulation of the two initiation strategies with an increasing S_{RSSI} in order to evaluate the following two numbers: $N_{RSSI < S_{RSSI}}$, that is how many times the RSSI is lower than S_{RSSI} , and $N_{\alpha^{(L)} > \alpha_T^{(L)}}$, that is how many times $\alpha^{(L)}$ is greater than $\alpha_T^{(L)}$; and ii) $P_r(H)$ estimation for each strategy. The probability is estimated with respect to the frequency approach. It can be thus evaluated as $P_r(H) = N_{RSSI < S_{RSSI}}/N_{tot}$ according to equation 1, whereas, for α -count strategies, it is $P_r(H) = N_{\alpha^{(L)} > \alpha_T^{(L)}}/N_{tot}$, according to equation 2, where N_{tot} is the total number of measurements.

The simulation is performed by populating the models with realistic values. Hence, RSSI values were obtained on the experimental field. In particular, we measured RSSI values with respect to the average case and the worst case. For the average case, we measured the RSSI of a typical user moving with his iPAQ around the lab, from room A to room B, from room C to room D, and so on, for several days. As for the worst case, we place the iPAQ between two rooms to force continuous handoffs due to lower RSSI values. The worst case scenario allows us to estimate the minimum guaranteed level of availability. Once these measurements have been performed, the initiation strategies have been simulated using RSSI measured values, and values of S_{RSSI} from 1 to 7. Simulation results are shown in figure 7. As we expected, the $P_r(H)$ is an increasing function of the S_{RSSI} threshold. The figure points out the benefits of our strategy, both in the average and in the worst case. In the average case, the α -count scheme produce respectively a 15% improvement on average, compared with the fixed threshold on RSSI scheme. This percentage increases to 45% on average in the worst case.

As far as performance experiments are concerned, we aim to i) measure the Round Trip Time (RTT) obtained between client and server applications using NCSOCKS both over Bluetooth and Wi-Fi; and ii) estimate the overhead of NCSOCKS library compared to standard transport primitives. Results are

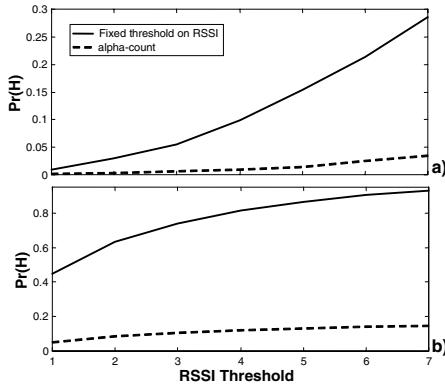


Fig. 7. Unavailability estimation: a) average case, b) worst case

Table 1. RTT and overhead results

	NCSOCKS	Standard Sockets	Overhead
Bluetooth	0.158 s	0.149 s	6.04 %
Wi-Fi	0.0056 s	0.0052 s	7.69 %

depicted in table 1. Measurements have been performed using UDP datagrams with 1000 bytes of payload and assuming a distance of 2 meters between mobile device and AP. It is worth noting that the RTT values we measured over the Wi-Fi channel are comparable with results obtained in [9], [11]. As table 1 shows, the overhead is quite acceptable, in that it is at most 7.7%.

5.3 Locationing System Tuning

The proposed α -count schema affects the *reaction time* of the initiation strategy. We define the reaction time T_r as the time within which the handover is triggered, once the mobile device reaches the cell boundary. T_r is a function of α -count parameters, that is $T_r = f(S_{RSSI}, \alpha_T, dec)$; hence, once the expected size of a cell is fixed, it is necessary to tune the α -count parameters in order to achieve a certain T_r when the boundaries are reached. We define an experimental procedure to tune the α -count parameters, with respect to the expected cell size and T_r . For each cell, the tuning process encompasses two steps: **i**) experimental evaluation of RSSI frequency distribution at the cell boundary, and **ii**) simulation of the α -count algorithm with the previous evaluated RSSI distribution in order to estimate T_r as a function of the triple $(S_{RSSI}, \alpha_T, dec)$. These steps are performed by the prober (see section 2.5). Once fixed T_r , it is possible to choose proper values for α -count parameters, which are sent to the Map Server by the prober, and stored in the topology map, for each AP.

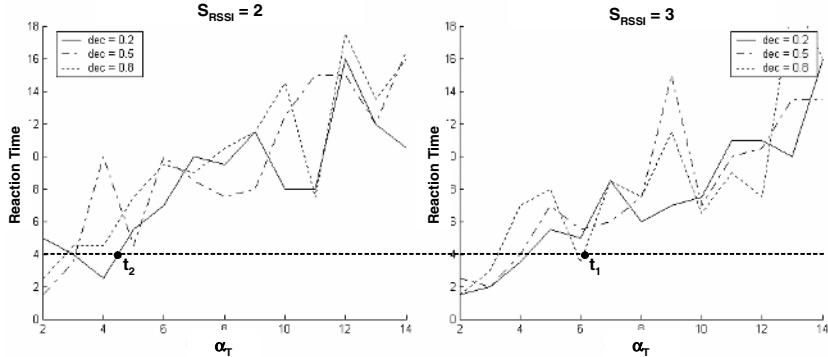


Fig. 8. α -count parameters tuning

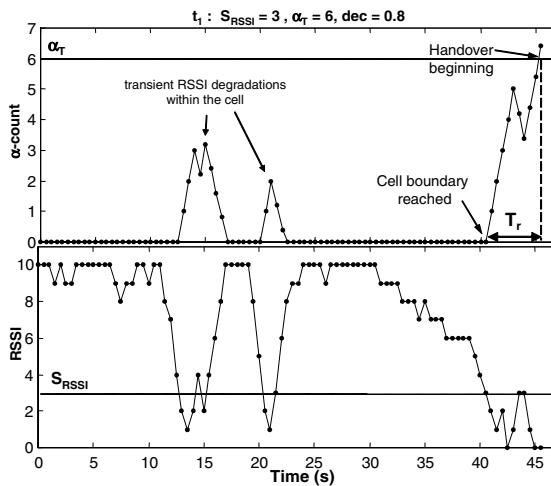


Fig. 9. Experimentally measured RSSI and related α -count behavior

To exemplify, let us consider the tuning of the Bluetooth cell B (see figure 6). According to the step i), we use the prober to capture RSSI values in several parts of the cell boundary region. As for step ii), α -count simulation results are depicted in figure 8, where T_r is reported as a function of α_T , dec , and S_{RSSI} parameters. Hence, once fixed the desired T_r (as an example $T_r = 4$ in the figure, which is emphasized by a dashed line), it is possible to determine different $t = (S_{RSSI}, \alpha_T, dec)$ triples that produce the expected T_r (for example both $t_1 = (S_{RSSI} = 3, \alpha_T = 6, dec = 0.8)$ and $t_2 = (S_{RSSI} = 2, \alpha_T = 4.5, dec = 0.2)$ triples can be used). Figure 9 shows the experimentally measured RSSI and the related α -count behavior obtained by using the t_1 triple for the parameters. As the figure points out, once reached the cell boundary (after 40s of experimentation), the experimental T_r is about 4.8 seconds, as expected.

6 Conclusions

This work presented a mobility management and communication architecture for nomadic environments, which aims to achieve connection and location awareness in spite of wireless network heterogeneity, while improving the availability level of connection. We presented the design of the proposed architecture using a pattern-oriented design approach in order to propose a general solution which can be adopted irrespective of the wireless technology being adopted. In order to improve the communication availability, we proposed a new mechanism, based on the α -count function, capable of discriminating transient RSSI degradations from persistent ones. This led to a substantial reduction of the number of unnecessary handoff procedures. In order to estimate availability improvements and to configure α -count parameters to tune the locationing service, we adopted an approach based on combined use of simulation and prototype-based measurements. Values for the simulation model parameters were extracted from direct measurements on the testbed, deployed over an heterogeneous wireless network, composed of Bluetooth and Wi-Fi appliances. Results demonstrated that the architecture is able to achieve availability improvements, thanks to the proposed α -count mechanism, at an acceptable performance penalty. We plan to use our communication library as a core building block of an enhanced distributed object computing middleware for nomadic environments, named ESPERANTO.

Acknowledgments. This work has been partially supported by the Italian Ministry for Education, University and Research (MIUR) in the framework of the FIRB Project “Middleware for advanced services over large-scale, wired-wireless distributed systems (WEB-MINDS), and by Regione Campania in the framework of “Centro di Competenza Regionale ICT”.

References

1. IEEE. *IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications;* 1999.
2. Bluetooth SIG. *Specification of the Bluetooth System - core and profiles v. 1.1,* 2001.
3. P. J. Mogowan, D. W. Suvak, and C. D. Knutson. *IrDA Infrared Communications: an Overview.* www.irda.org.
4. L. Kleinrock. Nomadicity: Anytime, Anywhere in a disconnected world. *Mobile Networks and Applications*, 1(1):351 – 357, December 1996.
5. O.B Akan and I.F. Akyildiz. ATL: An Adaptive Transport Layer Suite for Next-Generation Wireless Internet. *to appear in IEEE Journal on Selected Areas in Communications*, 2nd Quarter 2004.
6. J. E. Bardram. Applications of context-aware computing in hospital work-examples and design principles. *Proc. of the 19th ACM Symposium on Applied Computing (SAC 2004)*, March 2004.
7. S. Soucek, T. Sauter, and Koller. Impact of QoS parameters on internet-based EIA-709.1 control applications. *Proc. of the 28th Conf. of the Industrial Electronics Society, IEEE CS*, 2002, 2002.

8. M. Lampe, M. Strassner, and E. Fleisch. A Ubiquitous Computing Environment for Aircraft Maintenance. *Proc. of the 19th ACM Symposium on Applied Computing (SAC 2004)*, March 2004.
9. G. Bianchi, N. Blefari-Melazzi, M. Holzbock, Y. Fun Hu, A. Jahn, and Ray E. Sheriff. Design and validation of QoS aware mobile internet access procedures for heterogeneous networks. *Mobile Networks and Applications, Special Issues on Mobility of Systems, Users, Data and Computing*, 8(1):11–25, February 2003.
10. J. Tourrilhes and C. Carter. P-handoff: A protocol for fine grained peer-to-peer vertical handoff. *Proc. on the 13th IEEE Int. Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '02)*, 2002.
11. V. C. Zandy and B. P. Miller. Reliable network connections. *Proc. of the 8th Int. Conf. on Mobile Computing and Networking (MOBICOM '02)*, Sept. 2002.
12. Network Working Group, IETF. *IP mobility support, RFC 2002*, 1996.
13. E. Wedlund and H. Schulzrinne. Mobility support using SIP. *Proc. of the 2nd ACM Int. Workshop on Wireless Mobile Multimedia (WoWMoM'99)*, 1999.
14. J. Hightower and G. Borriello. Location systems for ubiquitous computing. *Computer*, 34(8):57–66, August 2001.
15. F. Gonzalez-Castano and J. Garcia-Reinoso. Bluetooth location networks. *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, Nov. 2002.
16. J. Hallberg and M. Nillson. Positioning with Bluetooth IrDA and RFID. Master's thesis, Uleå University of Technology, 2002.
17. C. A. Patterson, R. R. Muntz, and C. M. Pancake. Challenges in location aware computing. *IEEE Pervasive Computing*, 2(2):80–89, April-June 2003.
18. P. Reynolds. Mobility management for the support of handover within a heterogeneous mobile environments. *Proc. of First Int. Conf. on 3G Mobile Communication Technologies*, 27–29 March 2000.
19. A. M. Bin Ahamad and M. D. Bin Baba. Handover strategy for mobile wireless LAN. *Proc. on the 4th National Conf. on Telecommunication Technology, Malaysia, IEEE CS*, October 2003.
20. J. Lansford, A. Stephens, and R. Nevo. Wi-Fi (802.11b) and Bluetooth: Enabling coexistence. *IEEE Network*, pages 20 – 27, September/October 2001.
21. M. L. George, L. J. Kallidukil, and J. M. Chung. Bluetooth handover control for roaming system applications. *Proc. of the 45th Midwest Symposium on Circuits and Systems. MWSCAS-2002.*, August 2002.
22. A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni. Threshold-based mechanisms to discriminate transient from intermittent faults. *IEEE Trans. on Computers*, 49(3):230 – 245, March 2000.
23. D. Cotroneo, F. Cornevilli, M Ficco, S. Russo, and V. Vecchio. Implementing positioning services over an ubiquitous infrastructure. *Proc. of 2nd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (WSTFEUS 2004)*, May 2004.

Platform-Independent Object Migration in CORBA

Rüdiger Kapitz¹, Holger Schmidt², and Franz J. Hauck²

¹ Dept. of Comp. Sciences, Informatik 4, University of Erlangen-Nürnberg, Germany
`rrkapitz@cs.fau.de`

² Distributed Systems Laboratory, University of Ulm, Germany
`{holger.schmidt, franz.hauck}@uni-ulm.de`

Abstract. Object mobility is the basis for highly dynamic distributed applications. This paper presents the design and implementation of mobile objects on the basis of the CORBA standard. Our system is compatible to the CORBA Life-Cycle–Service specification and thus provides object migration between different language environments and computer systems. Unlike others, our Life-Cycle–Service implementation does not need vendor-specific extensions and just relies on standard CORBA features like servant managers and value types. Our implementation is portable; objects can migrate even between different ORBs. It supports object developers with a simple programming model that defines the state of an object as value type, provides coordination of concurrent threads in case of migration, and takes care of location-independent object addressing. Additionally we seamlessly integrated our implementation with a dynamic code-loading service.

Keywords: Object Migration, Platform Independency, CORBA, Life-Cycle Service, Value Types, Dynamic Loading of Code.

1 Introduction

One of the key features of object-based distributed programming environments like CORBA (Common Object Request Broker Architecture) is the transparent access to remote objects. The middleware infrastructure hides the distribution and the heterogeneity of the underlying computer hardware, operating system, and programming language. However, full access transparency is not always useful. Sometimes the true distribution of objects should be visible and controllable by applications. Examples are applications that explicitly move distributed objects for balancing load, for handling failures, and for minimizing communication overhead (e.g., mobile agents). For mobile objects the middleware system has to support state transfer and location-independent addressing of objects. Often, the support mechanisms are tightly woven into the middleware and therefore highly system dependent.

CORBA is ammended by the Life-Cycle–Service specification [1], which describes a service concept based on common design patterns to implement object

mobility and other life-cycle operations. Objects have to implement a special Life-Cycle interface that, among others, provides a `copy()` and a `move()` method to duplicate and migrate an object. Although the Life-Cycle–Service specification defines the general life-cycle process, it has certain shortcomings that lead to unnecessary burdens for application programmers and to system-dependent and incompatible implementations. We propose the design of a generic Life-Cycle–Service implementation, which is only based on common CORBA features and therefore vendor independent. Our prototype is implemented in Java, but can easily be ported to other CORBA-supported languages. Mobile objects can even migrate between different ORBs when those run an implementation of our service design.

For the application programmer, we provide a value-type–based state-transfer mechanism. This frees developers from writing their own state-exchange mechanisms for every mobile object. Furthermore we provide mechanisms for dynamic loading of code based on previous work [2]. Thus, the code for mobile objects needs not to be statically deployed. Finally our implementation encapsulates the coordination of life-cycle operations and frees the application programmer from location management. Our implementation either forwards requests or uses a lightweight location service.

The next section gives a brief introduction of the Life-Cycle–Service specification, its shortcomings, and existing implementations. In Section 3, we discuss different solutions for collecting and exchanging the state of a CORBA object. Section 4 describes the design of our generic Life-Cycle–Service implementation. The development process of a life-cycle object is illustrated by a simple example application in Section 5. Section 6 is devoted to performance evaluations. Finally, Section 7 discusses related work and Section 8 gives our conclusions.

2 CORBA Life-Cycle Service

CORBA is a standardized architecture defined by the Object Management Group (OMG) that allows programmers to create and access objects deployed in a distributed system. CORBA also specifies a set of CORBA services. These services represent optional ORB extensions and address general needs of CORBA applications. The Life-Cycle Service [1] is such a CORBA service, as it enables application-controlled mobility and other life-cycle operations. In the following sub-sections we will describe the core components of the Life-Cycle Service, discuss its weaknesses and shortcomings, and close with a brief overview of existing implementations.

2.1 Basic Functionality

Standard CORBA provides distribution transparency. With the help of an implementation repository migration of objects can also be made transparent [3]. A servant has to be registered at the implementation repository, which from this time on takes care that the servant remains accessible. This mechanism

allows restarting of server processes at different locations, but is not suited for application-controlled object mobility. Additionally, these implementation repositories are tightly woven into an ORB and its dependend POA¹ implementation [4].

In contrast, the Life-Cycle Service allows an application to control the distribution of objects, e.g., creating, removing, copying, and moving of objects. This is especially useful for mobile applications (e.g., mobile agents) and for the management of applications that needs to distribute objects across different platforms for non-functional reasons like scalability and fault-tolerance.

It is assumed that object creation is performed using factory objects. These can also be remote allowing for remote object creation. A factory is a CORBA object offering a method for creating new instances of a particular object type at a particular location. It is not specified how factories are requested to create a new object. This is left to the object developer as there can be different parameters required for different object types. However, the Life-Cycle-Service specification defines an IDL interface named **GenericFactory**. This interface contains a generic **create_object()** operation, which gets a set of criteria represented as sequence of name-value pairs in the IDL type **Criteria**. The Life-Cycle-Service specification gives hints on how to use criteria but does not define any standards. For a specific factory implementation they can be used to select the required object type, object capabilities, different object initializations, and even different locations. The latter can be accomplished by forwarding the creation request to a more specific factory object at a particular location depending on a particular criterion.

```
interface LifeCycleObject {
    LifeCycleObject copy( in FactoryFinder there ,
                         in Criteria the_criteria )
        raises( ... );
    void move( in FactoryFinder there ,
               in Criteria the_criteria )
        raises( ... );
    void remove()
        raises( ... );
};
```

Fig. 1. IDL specification of the **LifeCycleObject** interface

While object creation is handled by a factory, all other life-cycle operations are executed at the object itself. Therefore, an object supporting the Life-Cycle Service has to implement the **LifeCycleObject** interface (see Fig. 1). The **copy()** operation creates a copy of the object at some location. As a result, a reference to the newly created object is returned. The **move()** operation moves the object to another location; the **remove()** operation deletes the object.

¹ POA = Portable Object Adapter.

Both `copy()` and `move()` need some notion of location in order to place a copy or the object itself. The Life-Cycle Service specifies a `FactoryFinder` interface for objects representing an abstract location. Taking migration of an object as an example, Figure 2 shows the first phase of the interaction between object, factory finder and factory. For duplication of objects the scenario is almost identical.

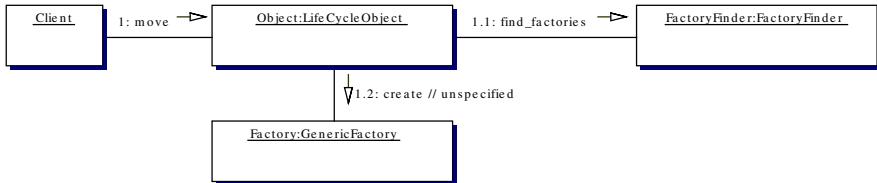


Fig. 2. First phase of object migration (UML collaboration diagram)

First a `move()` method is called on the object implementing the life-cycle interface. An instance of `FactoryFinder` is passed as parameter. The `move()` operation is supposed to ask the factory finder for a factory that finally can be used to create another instance of the original object at a certain location. In form of a key parameter, `move()` can ask for specific factory properties. The key is some sort of name-value pair that was originally introduced for naming objects [5]. Once again, the specification gives hints on how to use the key parameter but does not standardize anything. Anyway, from the key parameter the finder has to select a suitable factory.

The factory is a sub-type of interface `Factory` that remains unspecified². Thus, the `move()` implementation has to know the expected type and has to cast the factory reference to that type by a narrow operation. The factory can have type `GenericFactory` and the criteria set passed to `move()` can be used to influence the factory in creating the object. In the end, `move()` can create a new instance, transfer the state of the original object to the new one, and finally take care that the original object reference remains valid, now referring to the newly created object.

2.2 Open Issues and Shortcomings

The Life-Cycle Service is just a specification. Although the interfaces are specified in detail, the flow of control is just roughly described and implementation details are left to the object developer or the service provider. On one side, this allows for individual implementations of the specified interfaces, as the OMG deliberately underspecified certain issues in order to get them solved by an actual implementation. On the other side, it is likely that Life-Cycle-Service implementations become system dependent and incompatible.

² In fact `Factory` is just an IDL typedef to `CORBA::Object`, which has subtle differences to a sub-type.

There are a number of problems with the specification. First, there is no concept specified how the **FactoryFinder** locates existing factories. It is, however, possible to use a Naming Service to retrieve an object by using the key parameter as a name. In summary, the configuration of factories and factory finders is outside of the specification and has to be done by developers. Second, after migration of an object all references to this object should stay valid to maintain location transparency. Unfortunately, the precise procedure for solving this problem is left to the service implementers. Third, the service specification assumes that at each location the required code of the object servant is available. In dynamic environments with mobile objects, it would be preferable to be able to transfer and load code on demand. In a scenario with mobile agents, we cannot assume that the agent code is present at every possible location. Fourth, the most severe problem with the specification is that it does not provide any measures for state transfer. It is neither specified how to determine and gather the state nor how to do the transfer. Usually it is left to the object developer to write the corresponding code. Finally, the specification does not deal with any kind of coordination of concurrent threads. Multiple threads may invoke life-cycle and normal operations that in turn may interfere with state collection and transfer.

In total, we believe that there are too many unnecessary burdens left for the application developers. Additionally, the individual solutions of service developers to the above-mentioned problems make it hard to port life-cycle objects from one system to the other. Migration between different ORBs is usually impossible.

2.3 Implementations of the Life-Cycle Service

There are numerous commercial and non-commercial ORB implementations around that offer facilities and interfaces for application developers to copy or migrate CORBA objects. But either these solutions are platform-dependent like those provided by omniORB [6], ACE ORB (TAO) [7], or the Plug-In Model [8], or they even do not support the Life-Cycle-Service specification at all and provide a totally vendor-specific solution like VisiBroker [9]. None of these implementations addresses platform-independent migration of state and code in heterogeneous environments.

In [10], Peter and Guyennet propose a generic solution for object mobility in CORBA based on the Life-Cycle-Service specification. They focus on coordination of object access during and immediately before and after migrations to increase availability. The object state has to be described as an IDL structure that is used to generate special state-carrying objects with custom access methods. Forwarding is realized using tool-generated proxy objects on the client and the server side that use the CORBA Naming Service as location service. This imposes special actions on the client side. Furthermore, the implementation does not address the provision of platform-dependent code.

Choy et al. describe a CORBA environment supporting mobile agents based on mobile CORBA objects [11]. Based on the Life-Cycle Service and the Externalization Service a concept was mooted, but apparently not implemented.

3 State Transfer in Heterogeneous Environments

As described in Section 2.2, the migration of an object requires the transfer of its state. In homogeneous and more or less platform-independent environments like Java the execution environment may already provide serialization mechanisms. State transfer in CORBA is more challenging, since the state of an object may be transferred between different language environments, i.e. from C++ to Java or Cobol. In this case, language-dependent solutions will fail.

There should be a language-independent and fairly abstract transfer format. For example, it does not make sense to convert the content of a Java `Hashtable` object into a transfer format, as there are about 30 internal and implementation-dependent variables that can hardly be restored in a C++ implementation of that hash table. Instead, it is necessary to distinguish between the state of a particular object implementation and a more or less implementation-independent state that is essential for the object semantics. For a hash table the abstract state will only contain the stored key-value pairs. This abstract state can hardly be automatically identified; instead this has to be done by the developer. Finally, it is useful to define the abstract state in a format that can easily be converted into all supported programming languages so that object developers immediately can identify the transferable object state.

As already mentioned, the Life-Cycle Specification does not specify how to collect and transfer state. Instead, it suggests letting the developer use either a proprietary solution or the CORBA Externalization Service [12]. A proprietary solution may be appropriate if mobile objects move within a homogeneous environment as language-based serialization mechanism can be used. In case of heterogeneous language environments, the object developer needs to find an individual solution for transfer of state, which is likely to be complex and error-prone.

The CORBA Externalization Service was developed to support writing an object state into a data stream and reading that state back from a stream. Whereas this was basically designed for persistence the same concept can be used to support state transfer by shipping the externalized stream to another location and internalize that stream back into another object. Although the Externalization Service offers a common data format this approach has some serious drawbacks. The developer has to write his own marshalling and unmarshalling procedures that call the right operations of the Externalization service in the specific order. This has to be done for every language that is used. In principle, it would be possible to describe the abstract state in some language-independent format and automatically generate the marshalling procedures. However, to date there are no known tools for generating those procedures.

A promising and obvious approach is the description of the transfer state via IDL. Peter and Guyennet [10] used an IDL struct type and provided tool-generated wrapper objects with access methods. This is quite complex and the developer has to implement the invocations of those methods. Instead, we want to support a more generic approach of state transfer that unburdens the developer from calling serialization and deserialization methods at all. For state transfer, we propose IDL value types, a well-known part of the CORBA specification [13].

A value type is similar to an IDL struct, but it can also have methods much like CORBA objects. CORBA objects are declared with IDL interface types. Passing a CORBA object to a possibly remote method transfers the reference to this object (call-by-object-reference semantics). In contrast, passing value-type objects leads to a complete copy of the value type at the receiving side (call-by-value semantics). Like CORBA objects, value types also support inheritance.

Transfer of a value type is realized by transparent marshalling and unmarshalling of the state of the value-type object. In a heterogeneous system it is possible to rebuild value types implemented in one language in another one, e.g., from Java in C++.

```
interface Account{ ... };

valuetype AccountContainer supports Account {
    private float account_state;
};
```

Fig. 3. IDL value type declaration with the supported interface

Like CORBA objects, value types are able to support a specific IDL interface (see Fig. 3). This implies that methods specified in the supported interface are implemented in the value type. A value type supporting an interface can be activated at a POA, and is then remotely accessible. With activation a value type behaves as an ordinary CORBA object that can be passed by reference. Nevertheless it is also possible to pass the value type by value, creating a copy of the value-type object.

For state transfer, we are using value types that support a particular IDL interface. Object functionality has to be encapsulated in a value-type implementation. Thus, the public and private members of the value type represent the abstract state of the object, and the supported IDL interface represents the object's remote methods (cf. Fig. 3). Activated at a POA, the value type works as an ordinary CORBA object. In case of a state transfer, we just pass the underlying value type with call-by-value semantics to another location, which will marshal and unmarshal the necessary state. The object implementation is usually determined by factories registered at the ORB.

To sum up, value types are perfect candidates for implementing state transfer. Value types are well known to CORBA developers since they are part of IDL. Value types can implement CORBA objects and be values at the same time. They document the state and allow the IDL compiler to automatically generate all necessary serialization and deserialization procedures; developers do not have to program them any longer. In the next section we will show how value types can be used in conjunction with a specialized factory to design a generic Life-Cycle Service.

4 Design of a Generic Life-Cycle–Service

This section proposes our generic Life-Cycle Service based on value types and describes the specific implementation details.

4.1 Finding and Selecting an Appropriate Factory

The first step before actually copying or moving an object is the selection of an appropriate factory. The application controls the selection process by passing a factory finder and so-called *criteria* parameters to the life-cycle operation. For the management of multiple factories, we supply a basic factory finder, which will match the needs of most applications. It extends the specified interface by methods for registering and removing factories and other factory finders. The latter enables the common CORBA approach of federations to gain scalability and flexibility.

If a life-cycle object calls the `find_factories()` method, our factory finder looks for matches in the local factory registry. The specification proposes two possible types of factories: specific factories and generic factories. Specific factories support only one type of object; generic factories can support multiple types of objects. To determine if a generic factory is able to create a certain object type it provides a `supports()` method. The factory finder will collect matching factories from its local registry and from all registered factory finders. In turn, these finders can also manage other finders and so on, building a hierarchical federation of finders. After the finder has passed the matching factories, the life-cycle object has to select the appropriate factory based on the criteria parameters and additional object-specific requirements. As this is object-specific it is supposed to be implemented by the developer.

The selection process of the appropriate factory, however, is supported by the generic factory interface as explained in Section 2. A generic factory provides a `create()` method that takes two parameters: a key referencing the object type and a criteria parameter. If the object type is not known to the object or the criteria cannot be satisfied, an exception is thrown.

We provide two variants of generic factories: a single-type and a multi-type factory. The single-type factory can only instantiate a single object type but offers the possibility to check criteria special to this factory and object type. The multi-type generic factory represents a registry for single-type generic factories. As an additional benefit the multi-type factory can process general criteria before even asking other factories. This way, general criteria can be validated that apply to all factories managed by a multi-type generic factory (e.g., checking resource requirements).

For both types of generic factories we provide a basic implementation that only requires an object which implements our `CriteriaChecker` interface. This interface offers a single method `checkCriteria()`, which is executed at the beginning of each creation request. It throws an appropriate exception if the criteria requirements either could not be met or are simply invalid. If no exception is thrown, the creation process proceeds. The single-type factory represents simply a facade implementation of the basic factory, which is explained in detail in Section 4.3, with an extended interface. So criteria could be handed over to the creation methods.

4.2 Coordination of Life-Cycle Operations

During a life-cycle operation, the access to an object has to be coordinated and restricted. For consistency, all three life-cycle operations need exclusive access to

the object in the sense that all earlier invocations have terminated and all others are blocked until the life-cycle operation is executed. Implementing custom coordination mechanisms at the object level could ensure this, but would require deep understanding of application and life-cycle functionality. Therefore, we decided for a solution transparent to object developers.

The first implementation alternative is a specialized POA that controls the access to servants representing life-cycle objects, but a modified POA would be an unwanted ORB-specific extension. Another alternative is the usage of a POA manager, because it can control the state of a POA and block incoming calls via the `hold_requests()` method. Unfortunately, it turns out that this does not work as `hold_requests()` cannot be safely called from a life-cycle operation³. Alternatively, portable interceptors represent a central point in the ORB architecture where all incoming and outgoing calls can be caught and modified. A life-cycle object could be registered at a special interceptor at creation time. The interceptor can analyze invocations and take care of proper coordination. However, intercepting all incoming calls is very expensive, as it slows down every method invocation even of non life-cycle objects.

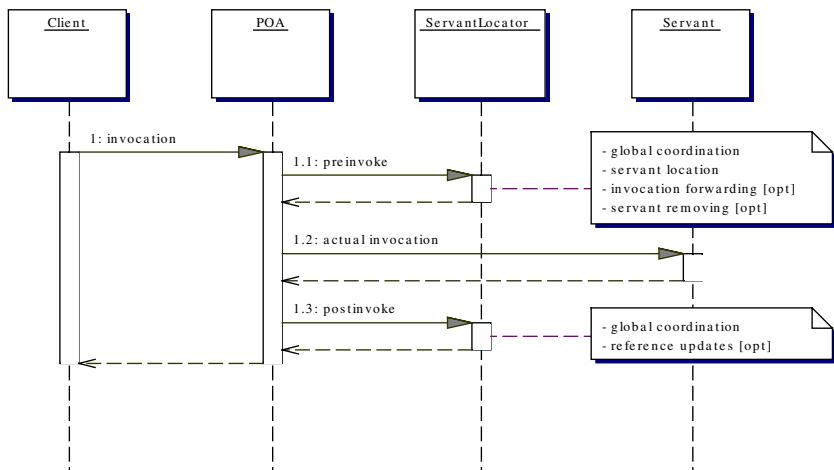


Fig. 4. Operating sequence of an incoming invocation (UML sequence diagram)

We decided in favor of a fourth alternative, a servant-locator-based approach. A servant locator is a special form of a servant manager that is responsible for the activation and management of servants. On every incoming call the POA notifies the servant locator by calling its `preinvoke()` method. The servant locator now has to locate or set up an appropriate servant and return it to the calling POA. After the method invocation on the servant, the POA calls the servant locator again by invoking the `postinvoke()` method. This triggers the locator to tear down the servant or to do other management tasks (Fig. 4).

³ The calling operation would try to wait for its own completion.

For the access coordination, we implemented a special servant locator that encapsulates the access management on behalf of the life-cycle objects. Every life-cycle object is registered on creation at the locator⁴ and owns a synchronized invocation counter, which is managed by the locator. If a method of a registered servant is called, the `preinvoke()` method is executed and the counter is incremented. After the actual invocation, the `postinvoke()` method decrements the counter of the servant. This way all pending calls are accounted.

A life-cycle operation can be detected by the servant locator because the method name of the servant invocation is passed to the `preinvoke()` method. If there are currently other pending calls, the life-cycle call is suspended until all others invocations were finished. Then, the life-cycle operation can be executed. Other incoming invocations—either normal or life-cycle operations—are suspended until the current life-cycle operation will have finished.

4.3 Creation of a Life-Cycle Object

The creation of a life-cycle object being supported by a special kind of factory is a key point in our design. As mentioned earlier, factories support the creation of objects at remote sites on behalf of a life-cycle operation. The factory design pattern is also very useful at creation time of an object. It encapsulates and hides setting up the environment for a life-cycle object and therefore reduces the programming effort for an application developer. Since these tasks are very similar for all life-cycle objects we implemented a general *base factory*. This factory only needs to know the name of the implementation classes to set up a new life-cycle object. The actual creation process has four steps:

1. Instantiation of a servant
2. Activation of the servant
3. Creation of a CORBA reference
4. Registration at a location service (optional)

The first step is straightforward. If one of several `create()` methods of the `BaseFactory` is called, the appropriate instances for the value type are created⁵. An explicit creation of a value-type object is only necessary if the object is completely new (e.g., by calling `create()`). In case of a migrated or copied object a value-type object that encapsulates the object state is passed to a `createFromValueType()` or `createCopyFromValueType()` method. Afterwards the value-type object is activated as a servant at the servant locator. In the next step a CORBA reference has to be generated. On initial creation of a life-cycle object a unique random object id is assigned. This id remains stable for the whole lifetime of the object even in context of `move()` operations. To ensure this and to free the application developer from unnecessary programming effort, every value-type object has to inherit from `MobileContainer`. This value type provides two

⁴ Our implementation also allows the registration of application-defined servant locators that are used as delegates from our locator.

⁵ In Java a separate Tie class is needed additional to the value-type implementation.

attributes, one for the object ID and another for the forwarding mechanisms. On creation of a new object, a new ID (UUID) is generated and set. In context of a `move()` operation the id can be read from the transferred value type. After the reference is generated, the optional registration at a location service takes place. This is covered in more detail in Section 4.5.

4.4 Dynamic Code Provision

Up to now, our implementation of a platform-independent Life-Cycle Service does not address the mobility of code. In dynamic environments, however, it is often required to transfer not only the state of an object but also the code implementing that object. The CORBA specification provides a code-base parameter for value types to dynamically load code on demand. Unfortunately, the specification suggests that the code base references directly the code of one or more implementations. This is sufficient if an object moves between homogeneous environments, but restricts flexibility if the value type is moved between different language environments.

We provide a special generic multi-type factory that is based on our *Dynamic Loading Service (DLS)* [2]. This service offers the dynamic loading of platform-dependent code on demand for arbitrary functionalities in an ORB-independent fashion. If the generic factory is asked whether a certain type is supported or if a creation of a previously unknown type is requested, the DLS will be queried. If an appropriate object implementation for the current platform exists, the DLS will dynamically load the code and instantiate the associated factory.

For using the DLS, there needs to be a DLS implementation that can be plugged into the local ORB. As DLS is portable to different ORBs this is not a problem. If a DLS is not available, object developers have to link the necessary code into the local system, as in most other Life-Cycle–Service implementations.

4.5 Forwarding and Locating

The reference to a life-cycle object should be valid for its entire lifetime even after migration. Our implementation addresses this problem in two ways: We provide a forwarder-based approach, where the servant locators at previous locations will cooperate in locating objects, and a location-service-based approach, where such a service simply keeps track of the actual object location.

The forwarder-based approach is the default, as it requires no additional preparations and services. The migration of a life-cycle object is initiated by calling the `move()` method. Inside the `move()` method the appropriate factory is selected and invoked. The factory returns a reference pointing to the new location of the object. This new location has to be announced to our servant locator. This is done indirectly by setting the `location` field of the base value-type `MobileContainer`, which every life-cycle object has to extend. After the `move()` operation, this value can be read by the locator inside the `postinvoke()` method. If the move operation fails due to unavailability of an appropriate factory, the location field will not be modified. The servant locator can detect this and the local object remains active.

The location value is registered in a special forwarding table. As already described in Section 4.2, after a life-cycle operation all waiting calls are resumed. Instead of actually invoking the methods on the local object which has moved, the calls are forwarded to the new location. This can be easily done, as the `preinvoke()` method offers a way to throw a forwarding exception with the new object reference. The client-side ORB handles this exception transparently by reissuing the request to the new location. Further requests are automatically forwarded to the new location as the ORB remembers location changes. This approach is very simple and has almost no additional overhead (just maintaining the forwarding table). It requires, however, that the object adapters at previous locations stay up until the object is finally removed. Another downside is a potentially long forwarding chain. Binding to a very early address could cause the first request to be forwarded many times tracking down the route of the object to the current location. More severe is that this method fails if only one of the hosts in the chain crashes or is down for some reason.

To avoid those drawbacks we implemented a simple location service as an additional solution. The key idea is to replace the references provided by our factory implementations. Instead of returning the actual object reference, it is modified to refer to a location service first. This way the location service receives the first invocation after an object is bound and forwards it to the actual location. In order to seamlessly integrate such a service into our implementation, it provides a CORBA management interface for registering and updating locations of life-cycle objects. On creation of an object the factory has to register the object at the location service and modify the returned reference to refer to the location service.

The service itself is also implemented as a servant manager, usually in a separate server process. Instead of locating or setting up the requested servant, the servant manager simply throws a forward exception referring to the actual location of the object. As previously noted this exception is transparently handled by the client-side ORB and the request is invoked on the actual location of the object. If the object moves to another location the factory registers the new location of the object at the service. After a move operation, the old servant locator throws a forwarding exception referencing again the location service which forwards the request to the actual location of the object. If the object is moved and the previous server is no longer accessible, the client-side ORB will fall back to the initial object reference also referring to the location service, which by then knows the new location of the object. To avoid a single point of failure for all life-cycle objects and for scalability reasons, our implementation is able to use multiple location services.

5 Example Application

In this section we demonstrate the implementation of our Life-Cycle Service by a simple application. For the development of a life-cycle object, the following steps have to be performed:

1. Development of the IDL description of the object interface
2. Development of the state description by defining an IDL value type that implements the object interface
3. Implementation of the value type
4. Instantiation of the object with our `BaseFactory`

Our example is an `Account` object described in IDL, which implements simple bank-account functionality. As shown in Fig. 5, this `Account` interface has to inherit from `LifeCycleObject`. The specified methods are object-dependent and implement the needed account functionality.

```
interface Account : ::CosLifeCycle::LifeCycleObject {
    float getBalance();
    void deposit( in float value );
    void withdraw( in float value );
};

valuetype AccountContainer :
    ::org::aspectix::services::lcs::MobileContainer
    supports Account {
    private float balance; ...
};
```

Fig. 5. `Account` interface and the corresponding value type (IDL)

In a next step the IDL description of the value type actually implementing the appropriate object functionality has to be specified. In this value-type declaration also the state has to be specified using private or public data members. The value type has to inherit from `MobileContainer` as described in Section 4.3. Furthermore, it also has to support the previously specified IDL interface. In the example, we declared a value type supporting the `Account` interface (Fig. 5).

The private variable `balance` implements the actual state of the `Account` object, namely the balance information. This state will be transparently transferred via the call-by-value semantics of the value type in case of a move or copy operation (cf. Section 4).

From IDL an abstract `AccountContainer` class is automatically generated. We have to implement a concrete class `AccountContainerImpl` containing all methods of the interface and value type. Of course, the `LifeCycleObject` methods have to be implemented, too. As our Life-Cycle Service takes care of coordination and request forwarding, the actual implementation is rather simple. As we cannot show examples due to length restrictions, we roughly sketch their implementation: In the `move()` and `copy()` method, the developer just has to call `find_factories()` at the `FactoryFinder` and select the intended factory. Finally, the creation method on the factory has to be called. Thus the developer can influence the process of selecting an appropriate factory. The code of the `remove()` method just has to decide whether the object can be deleted or not. If no exception is raised by the operation,

the object will be automatically removed by our servant locator after the execution of this method returns. Within this method the developer is able to do application-specific tasks like deleting external files, etc.

All objects have to be created with our `BaseFactory`. This ensures the necessary POA policies, transparently involves a location service, and reduces development efforts. The value-type object might be created directly in the `BaseFactory` or it might be created and passed to the factory's `create()` method. To run our example application, a `FactoryFinder`, the factories and if necessary a location service have to be started on different machines.

6 Measurements

As our implementation delays calls due to coordination efforts even in cases of no migration, we performed different measurements for estimating the performance penalty of our approach. In another series of measurements we compare different methods of state transfer. All measurements were performed on Intel Xeon 2.4 GHz machines having 2 GB of RAM. We used Java JDK 1.4 and JacORB version 2.2. Effects caused by just-in-time compilation and other run-time optimizations in the JVM have been smoothed out.

6.1 Overhead of Migratable Objects

We first compare the implementation of a CORBA object using a value type with the standard implementation of a CORBA object based on the generated skeleton code. The measurements were performed on two different ORB implementations, JacORB [4] and Sun's built-in ORB from the JDK.

Table 1 shows the results: The first line shows the time needed for a local call sent to a standard CORBA object. The next line presents the time needed for an object implemented by the value-type approach. In the following lines the difference to the standard case is shown, e.g., on JacORB the overhead per call is about 130 ns or 2.3%. In the last three lines we added our servant locator, which has to detect life-cycle operations, coordinate invocations and maintain forwarding if necessary.

Table 1. Difference of time needed for a call using JacORB and SUN ORB

Variant	JacORB	SUN ORB
Standard Skeleton	5.67 μ s	4.74 μ s
Activated Value-Type	5.80 μ s	4.78 μ s
Overhead compared to Standard	0.13 μ s 2.3 %	0.04 μ s 0.8 %
Life-Cycle Object (Value Type and Locator)	9.65 μ s	11.88 μ s
Overhead compared to Standard	3.98 μ s 70 %	7.14 μ s 151 %

Our measurements did invoke ordinary operations but still the detection of life-cycle calls and the check for a necessary forward to another location takes considerable time. The implementation uses a Java `Hashtable` that may be the bottleneck. However, we have not yet optimized the implementation for performance.

The measurements based on the Sun ORB show similar results. Compared to the JacORB the value type is much faster, and the locator takes considerably more time. As our locator is exactly the same the additional time is consumed inside of the Sun ORB, but we did not yet investigate where.

6.2 State Transfer

In the next step we compared different methods of state transfer: Java serialization, IDL struct and value type. In all three scenarios we use JacORB and a CORBA remote invocation between a client and a CORBA object. Client and server systems are connected via switched 100 MBit Ethernet LAN. The transferred state contains two long values, two strings containing in total 13 characters and a small octet sequence about 26 bytes.

Table 2. Difference of time needed for state transfer (using JacORB)

Variant	JacORB
Java Serialization	880 μ s
IDL Struct	960 μ s
Overhead compared to Serialization	80 μ s 9.1 %
IDL Value Type	1,150 μ s
Overhead compared to Serialization	270 μ s 31 %
Overhead compared to IDL Struct	190 μ s 20 %

The first test uses Java serialization on the client side to convert the object state into a byte array that is passed by value to the server where it is deserialized. Note that we used a CORBA method call here to pass the serialized data. The measurement serves for comparison with the CORBA-based state transfer techniques and basically shows how much overhead can be saved in a homogeneous environment. The second and third measurements show the invocation time when transferring state by using an IDL struct and a value type. Both objects are reconstructed at the server side without further computation. In both cases, the code is already deployed. Table 2 shows the complete measurements including overheads compared to serialization.

The overhead of a value type compared to a struct is relatively large because JacORB uses reflection to instantiate the particular implementation class of the value type whereas in case of a struct the implementation class is hard-coded into the demarshalling operation. Still there are possible optimizations that we have

not yet investigated in detail. The value-type approach has the additional advantage that the state is already in place at the member components of the value type. With the struct approach the application usually has to access the data in the struct. Using Java serialization is only slightly more efficient and cannot cope with heterogeneous platforms. Of course the precise performance figures depend on the size of the state being transferred. We expect that the larger the state the less dominant the overhead of a value type compared to a struct will be. On the other hand, Java serialization will always be more efficient, but cannot deal with heterogeneous environments.

7 Related Work

As already stated in Section 2.3, there is no implementation of the CORBA Life-Cycle Service that addresses the platform-independent object migration of state and code in heterogeneous environments as our proposed solution does. Only Peter and Guyennet offer in [10] a generic solution for object mobility based on the Life-Cycle Service. However, their solution requires special client-side proxy objects that forward requests. This way, a client has to be aware of life-cycle objects. The object state has to be described as an IDL structure that is used to generate special state-carrying objects with custom access methods. Our value-type-based solution also requires the declaration of the state via IDL, but is more convenient for developers since they need not to call the access functions of custom objects on serialization and deserialization of an object. Furthermore, the implementation of Peter and Guyennet does not address the provision of platform-dependent code.

In the past, a lot of mobile agent systems have been developed, like MOA [14], Mole [15] or Aglets [16], to name a few. But they all provide only mechanisms for migration in homogeneous environments. An Agent Transport Service (ATS) was specified in [17]. There, the Life-Cycle Service was considered, but finally sorted out in order to support lightweight agents. All migration methods are offered entirely by the platform; the agent developers do not have to write any code for this purpose. The Object Management Group (OMG) developed a standard for agent communication: the Mobile Agent System Interoperability Facility (MASIF) [18]. The CORBA Life-Cycle–Service was considered, but as many agent platforms are not based on CORBA, they created their own methods for migration. As an example, many systems use Java Serialization, which can be deployed for homogenous environments only.

In [19] a platform-neutral approach of agent migration is presented. Instead of transferring the code just a blueprint is transmitted. This is possible by assuming that an agent consists of different components. For creating such an agent an *Agent-Factory* is specified that creates an executable agent consisting of the right components from such a blueprint. Migration is thus based on transferring the blueprint and the state of an object. The same approach could be layered on top of our Life-Cycle–Service implementation by structuring a complex application as a set of objects each representing a component that are moved together as it is proposed by the CORBA Relationship Service [20].

Finally the usage of value-type objects in our Life-Cycle-Service implementation bares some similarities with the functionality of the Freeze Evictor of the ICE middleware [21]. There the ICE equivalent of a CORBA value type is used to store objects in persistent storage and to access them as remote objects at the same time.

8 Conclusions

We presented a platform-independent implementation of the CORBA Life-Cycle-Service. Although our prototype is implemented in Java it can be easily ported to all CORBA-supported languages and then be used in different ORBs, as it is based on standard CORBA and does not need any ORB-specific extensions.

For state transfer even in heterogeneous environments, we introduced value types. The state can easily described in IDL, and developers do not need to implement state transfer routines. Furthermore, we also provided a solution to code provision in heterogeneous environments. Based on previous work, we offer an optional multi-type-supporting generic factory that loads platform-specific code on demand. This allows the dynamic instantiation of previously unknown objects. Finally our implementation presents flexible mechanisms to provide persistent object references in context of object mobility.

Apart from the fact that the current implementation has already reached a mature state, there are still possible extensions. We currently do not address how to secure the life cycle operations. This can be achieved by doing authentication either at the transport level with the Secure Sockets Layer protocol or at the object level. We also plan a service implementation in C++ and Python, but do not expect any implementation problems. Finally, on top of the current service implementation a mobile agent facility could be established. This would allow for agents that switch the implementation language while moving.

References

1. Object Management Group (OMG). Life Cycle Service Specification. OMG Document formal/2002-09-01, 2002.
2. R. Kapitza and F.J. Hauck. DLS: a CORBA service for dynamic loading of code. In *Proceedings of the OTM Confederated International Conferences*, Sicily, Italy, 2003.
3. M. Henning. Binding, Migration, and Scalability in CORBA. *Communications of the ACM special issue on CORBA*, 41:67–71, 1998.
4. A. Bendt et al. JacORB 2.2 Programming Guide, 2004.
5. Object Management Group (OMG). Naming Service Specification. OMG Document formal/2004-10-03, 2004.
6. S.-L. Lo D. Grisby and D. Riddoch. The omniORB version 4.0 User's Guide, 2004.
7. D.C. Schmidt. Real-time CORBA with TAO (The ACE ORB), May 2004.
8. C. Linnhoff-Popien and T. Haustein. Das Plug-In-Modell zur Realisierung mobiler COBRA-Objekte. In *Kommunikation in Verteilten Systemen*, pages 196–209, 1999.
9. Borland Inprise. VisiBroker for C++ 4.5 - Programmers Guide. Technical report, 2001.

10. Y. Peter and H. Guyennet. Object mobility in large scale systems. *Cluster Computing*, 3(2):177–185, 2000.
11. Breust Choy and Magedanz. A CORBA Environment Supporting Mobile Objects. Technical Report White Paper Draft Version 1, IKV++ GmbH, 1999.
12. Object Management Group (OMG). Externalization Service Specification. OMG Document formal/00-06-16, 2000.
13. Object Management Group (OMG). Common Object Request Broker Architecture: Core Specification, 2004.
14. W. LaForge D.S. Milojicic and D. Chauhan. Mobile Objects and Agents (MOA). In *4th USENIX Conference on Object-Oriented Technologies and Systems*, pages 179–194, Santa Fe, New Mexico, 1998.
15. J. Baumann M. Strasser and F. Hohl. Mole: A Java based mobile agent system. *Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems*, 1997.
16. D.B. Lange and M. Oshima. Programming and Deploying Java Mobile Agents Aglets, 1998.
17. C.A. Mendez and M. Mendes. Agent migration issues in CORBA platforms. In *The Fourth International Symposium on Autonomous Decentralized Systems*, pages 332–335, Tokyo Japan, 1999. IEEE.
18. D.S. Milojicic et al. MASIF: The OMG Mobile Agent System Interoperability Facility. In *Mobile Agents: Second International Workshop, MA ’98*, volume 1477/1998, page 50, Stuttgart, Germany, 1998. Springer LNCS. 1477.
19. F.M.T. Brazier et al. Agent factory: generative migration of mobile agents in heterogeneous environments. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 101–106, Madrid, Spain, 2002. ACM Press.
20. Object Management Group (OMG). Relationship Service Specification. OMG Document formal/2000-06-24, 2000.
21. The Internet Communications Engine (ICE), 2005.

Author Index

- Abdelmoty, Alia I. II-1466
Aberer, Karl, I-466, II-1243
Ahmad, Khurshid II-1330
Aldred, Lachlan II-1015
Alferes, José Júlio II-1553
Amador, Ricardo II-1553
An, Yuan II-1152
- Babaoglu, Ozalp I-612
Bacon, Jean I-366
Baker, Seán I-631
Balasubramaniam, Sasitharan I-846
Balasubramanian, Jaiganesh II-978
Barros, Roberto S.M. II-1381
Batista, Thaís II-1133
Bębel, Bartosz II-1347
Bender, Matthias I-310
Bensaber, Djamel Amar II-1640
Benslimane, Sidi Mohamed II-1640
Bessani, Alysson Neves, I-662, I-680
Beugnard, Antoine II-997
Bittner, Sven I-148
Blair, Gordon I-732
Blomqvist, Eva II-1314
Bontas, Elena Paslaru II-1296
Borgida, Alex II-1152
Borusch, Daniel I-680
Bosc, Patrick I-256
Briot, Jean-Pierre I-813
Brown, Ross I-94
Buchanan, George I-484
- Cacho, Nélia II-1133
Cappiello, Cinzia II-1535
Ceri, Stefano I-20
Cerqueira, Renato II-923
Chatti, Mohamed Amine II-1206
Cheang, Chan Wa II-1416
Chebbi, Issam I-112
Chen, David I-576
Cheong, Taesu I-557
Cinque, Marcello I-882
Conrad, Stefan I-539
Costa, Antonio Theophilo II-923
- Cotroneo, Domenico I-882
Coulson, Geoff I-732
Courtenage, Simon I-385
- da Silva, Paulo Salem II-1500
da Silva Fraga, Joni, I-662, I-680
de Beer, H.T. I-130
de Melo, Ana Cristina Vieira II-1500
De Meo, Pasquale I-329
Deng, Gan II-978
de Oliveira Valente, Marco Túlio II-1115
de Rijke, Maarten II-1432
Desai, Siddharth I-780
Deters, Ralph II-1097
Ding, Xiaoning II-1034
Dobson, Simon I-631
dos Santos, Hélio L. II-1381
Dumas, Marlon II-1015
- Eder, Johann I-502
Elbaum, Sebastian II-1065
Endler, Markus II-923
- Fankhauser, Peter II-1225
Fasli, Maria II-1571
Fekete, Alan I-40
Felber, Pascal II-1083
Ferdean, Corina I-796
Francalanci, Chiara II-1535
Freisleben, Bernd II-1046
Fu, Gaihua II-1466
- Gal, Avigdor I-402
Garcia-Haro, J. I-715
Garcia-Sánchez, Antonio-Javier I-715
Garcia-Sánchez, Felipe I-715
Gekas, John II-1571
Gergatsoulis, Manolis II-1188
Gillam, Lee II-1330
Giunchiglia, Fausto I-347
Goebel, Vera II-1365
Gokhale, Aniruddha II-978
Golze, Sebastian I-646
Gong, Zhiguo II-1416

- Gray, Alasdair J.G. I-420
 Greenfield, Paul I-40
 Gruszczynski, Paweł II-960
- Hadjali, Allel I-256
 Haïk, Grègory , I-813
 Halepovic, Emir II-1097
 Hauck, Franz J. I-900
 Hauswirth, Manfred, I-466, II-1243
 He, Yanxiang II-1588
 Heizmann, Jörg II-1261
 Henricksen, Karen I-846
 Herre, Heinrich II-1398
 Hidders, Jan I-220
 Hinze, Annika, I-148, I-484
 Hou U, Leong II-1416
 Huang, Tao II-1034
 Huhns, Michael I-453
 Hung, Edward I-1
- IJzereef, Leonie II-1432
 Indulska, Jadwiga I-846
 Iyer, Karthik I-453
- Jacobsen, Arno I-612
 Jaeger, Michael C. I-646
 Jang, Julian I-40
 Jarke, Matthias II-1206
 Jin, Beihong II-1034
 Jones, Christopher B. II-1466
 Jørgensen, J.B. I-22
- Kabilan, Vandana I-77
 Kammüller, Reiner II-1046
 Kamps, Jaap II-1432
 Kangasharju, Jaakkko I-274
 Kang, Dazhou II-1588
 Kang, Myong II-1483
 Kapitza, Rüdiger I-900
 Katsaros, Panagiotis II-941
 Kedad, Zoubida I-166
 Kementsietsidis, Anastasios I-292
 Kensche, David II-1206
 Kiani, Ali I-439
 Kim, Anya II-1483
 Kim, Youngil I-557
 Kiringa, Iluju I-292
 Kuo, Dean I-40
 Kutvonen, Lea I-593
 Kwasnikowska, Natalia I-220
- Lassen, K.B. I-22
 Leao, Diana Campos II-1115
 Lee, Minsoo II-1629
 Lee, Sung-Young II-1615
 Lee, Young-Koo II-1615
 Lehmann, Marek I-502
 Lehti, Patrick II-1225
 Lilis, Pantelis II-1188
 Linnemann, Volker I-613
 Li, Yanhui II-1588
 Liu, Bixin I-763
 Loebe, Frank II-1398
 Löser, Alexander II-1261
 Loyall, Joe I-612
 Lu, Jianjiang II-1588
 Lung, Lau Cheuk, I-662, I-680
 Luo, Jim II-1483
- Maciel, Paulo R.M. II-1381
 Mahleko, Bendick I-18
 Makpangou, Mesaac I-796
 Malki, Mimoun II-1640
 Masud, Md. Mehedi I-292
 Matougui, Selma II-997
 May, Wolfgang II-1553
 McFadden, Ted I-846
 Meersman, Robert II-1605
 Metso, Janne I-593
 Michel, Sebastian I-310
 Midonnet, Serge I-698
 Mühl, Gero I-646
 Munthe-Kaas, Ellen II-1365
 Mylopoulos, John II-1152
- Natarajan, Balachandran II-978
 Nepal, Surya I-40
 Neuhold, Erich I-18
 Ngoc, Kim Anh Pham II-1615
 Niederée, Claudia I-18
 Nutt, Werner I-420
- Oanea, Olivia I-183
 Oey, Mulyadi II-1065
 Osinski, Stanislaw II-960
- Paal, Stefan II-1046
 Pahl, Claus II-1170
 Paik, Hye-young I-94
 Pan, Jeff Z. II-1279
 Papadopoulos, Filippos I-864

- Papapetrou, Odysseas I-310
 Parsons, Jeff II-978
 Pavon-Mariño, P. I-715
 Pernici, Barbara II-1535
 Pietzsch, Dominik I-613
 Pitoura, Evaggelia I-864
 Pivert, Olivier I-256
 Popfinger, Christopher I-539
 Porto, Fabio II-1623
 Puder, Arno I-780
- Quattrone, Giovanni I-329
 Queinnec, Christian I-813
 Quix, Christoph II-1206
- Rashkovits, Rami I-402
 Reichert, Manfred I-59, I-238
 Rinderle, Stefanie I-59, I-238
 Rissee, Thomas I-18
 Rosa, Nelson S. II-1381
 Rossi, Pablo I-828
 Ruokolainen, Toni I-593
 Russo, Stefano I-882
 Ryan, Caspar I-828
- Sanderson, Norun II-1365
 Schiely, Marc II-1083
 Schlangen, David II-1296
 Schmidt, Douglas C. II-978
 Schmidt, Holger I-900
 Schrader, Thomas II-1296
 Schuhart, Henrike I-613
 Schweer, Andrea I-484
 Schwering, Angela II-1449
 Shi, Dianxi I-763
 Shi, Tony I-40
 Shin, Hyoseop II-1629
 Shiri, Nematollaah I-439
 Shvaiko, Pavel I-347
 Sidorova, Natalia I-183
 Silva, Rodrigo Palhares II-1115
 Sivaharan, Thirunavukkarasu I-732
 Skobeltsyn, Gleb II-1243
 Srisa-an, Witawas II-1065
 Sroka, Jacek I-220
 Subrahmanian, V.S. I-1
 Sun, Chengzheng I-576
 Sun, David I-576
 Swedrzynski, Andrzej II-960
- Tam, Audrey II-1517
 Tarkoma, Sasu I-274
 Tata, Samir I-112
 Tempich, Christoph II-1261
 ter Hofstede, Arthur H.M. II-1015
 Terracina, Giorgio I-329
 Thom, James A. II-1517
 Thomopoulos, Rallou II-1596
 Tirelo, Fabio II-1115
 Traversat, Bernard II-1097
 Tyszkiewicz, Jerzy I-220
- Udrea, Octavian I-1
 Ursino, Domenico I-329
- Van den Bussche, Jan I-220
 van der Aalst, Wil M.P. I-22,
 I-130, II-1015
 van Dongen, B.F. I-130
 van Hee, Kees I-183
 Vassiliadis, Panos I-864
 Vignéras, Pierre I-750
 Vu, Le-Hung I-466
- Wang, Huaimin I-763
 Wang, Shenghui II-1279
 Wang, Yufeng I-763
 Weber, Barbara I-59
 Weikum, Gerhard I-310
 Wild, Werner I-59
 Williams, Steven I-385
 Wombacher, Andreas, I-18, I-520
 Wrembel, Robert II-1347
- Xia, Steven I-576
 Xu, Baowen II-1588
 Xue, Xiaohui I-166
- Yatskevich, Mikalai I-347
 Yoneki, Eiko I-366
 Yu, Deng I-1
 Yu, Jonathan II-1517
 Yu, Zhiwei I-202
- Zarras, Apostolos I-864
 Zdravkovic, Jelena I-77
 Zhang, Li I-202
 Zhang, Xin II-1034
 Zhao, Gang II-1605
 Zlatev, Zlatko I-520