

Sören Auer
Oscar Díaz
George A. Papadopoulos (Eds.)

LNCS 6757

Web Engineering

11th International Conference, ICWE 2011
Paphos, Cyprus, June 2011
Proceedings



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

Alfred Kobsa

University of California, Irvine, CA, 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

TU Dortmund University, Germany

Madhu Sudan

Microsoft Research, Cambridge, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbruecken, Germany

Sören Auer Oscar Díaz
George A. Papadopoulos (Eds.)

Web Engineering

11th International Conference, ICWE 2011
Paphos, Cyprus, June 20-24, 2011
Proceedings

Volume Editors

Sören Auer
Universität Leipzig, Institut für Informatik
Abt. Betriebliche Informationssysteme
Postfach 100920, 04009 Leipzig, Deutschland
E-mail: auer@uni-leipzig.de

Oscar Díaz
Universidad del País Vasco
Euskal Herriko Unibertsitatea
Departamento de Lenguajes y Sistemas Informáticos
Facultad de Informática
Po M. Lardizabal, 1, 20018 San Sebastián, Spain
E-mail: oscar.diaz@ehu.es

George A. Papadopoulos
University of Cyprus, Department of Computer Science
POB 20537, CY-1678 Nicosia, Cyprus
E-mail: george@cs.ucy.ac.cy

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-22232-0 e-ISBN 978-3-642-22233-7
DOI 10.1007/978-3-642-22233-7
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011930445

CR Subject Classification (1998): H.3, H.4, I.2, C.2, D.2, H.5, J.1

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

© Springer-Verlag Berlin Heidelberg 2011

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.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Web engineering strives to accomplish the wish of an accessible, universal and affordable virtual platform where users and applications can smoothly interact. To this end, Web engineering systematically applies the knowledge of a large range of topics from software engineering to social sciences.

This volume contains the proceedings of the 11th International Conference on Web Engineering (ICWE 2011), which was held in Paphos, Cyprus, in June 2011. In Ovid's narrative, Paphos was Pygmalion's son. Pygmalion was a Cypriot sculptor who carved a woman out of ivory so realistically that he fell in love with the statue. Although perhaps not to the extent of falling in love, pioneers also exhibit passion about what they do. This makes Paphos a perfect place for passionate researchers and practitioners to meet.

The ICWE conferences represent first-class forums for the Web engineering community. ICWE is endorsed by the International World Wide Web Conference Committee (IW3C2) and the International Society for Web Engineering (ISWE). Previous editions took place at Vienna (Austria), San Sebastián (Spain), Yorktown Heights, NY (USA), Como (Italy), Palo Alto, CA (USA), Sydney (Australia), Munich (Germany), Oviedo (Spain), Santa Fe (Argentina) and Cáceres (Spain).

This year's call for papers attracted a total 90 submissions from 37 countries. Papers topics cover a broad range of areas, namely, the Semantic Web, Web services, mashups, Web 2.0, Web quality, Web development, etc. Submitted papers were reviewed by at least three reviewers from the Program Committee comprising 67 experts in Web engineering. Based on their reviews, 22 submissions were accepted (24% acceptance rate). In addition, 13 posters, 3 demos, and 4 tutorials were also part of the conference program. The conference hosted keynotes by Stephano Ceri (Politecnico di Milano) and Tim Furche (Oxford University). Last but not least, seven co-located workshops offered a venue for specialists to discuss on-going research and devise future research agendas.

Such an eventful program would not have been possible without the involvement of a large number of people and institutions. We would like to express our gratitude to the University of Cyprus as the local organizer. We are also indebted to Bebo White, Martin Gaedke, and Geert-Jan Houben who acted as liaison to the IW3C2, ISWE, and ICWE Steering Committee, respectively.

We are grateful to the Chairs of the different tracks, namely, Andreas Doms, Nora Koch, Andreas Harth, Cesare Pautasso, Steffen Lohmann, Axel Ngonga, Vicente Pelechano, Peter Dolog, and Bernhard Haslhofer. Finally, a special thanks goes to all the reviewers, the contributors, and the attendees, who substantially contributed to making ICWE 2011 a success.

May 2011

Sören Auer
Oscar Díaz
George A. Papadopoulos

Organization

Program Committee

Silvia Abrahao	Universidad Politecnica de Valencia, Spain
Sören Auer	Universität Leipzig, Germany
Fernando Bellas	University of A Coruna, Spain
Boualem Benatallah	University of New South Wales, Australia
Mária Bieliková	Slovak University of Technology in Bratislava, Slovakia
Davide Bolchini	Indiana University, USA
Athman Bouguettaya	CSIRO, Australia
Marco Brambilla	Politecnico di Milano, Italy
Jordi Cabot	INRIA-École des Mines de Nantes, France
Coral Calero	Universidad de Castilla-La Mancha, Spain
Fabio Casati	University of Trento, Italy
Sven Casteleyn	Universidad Politécnica de Valencia, Spain
Key-Sun Choi	Semantic Web Research Center, KAIST, South Korea
Richard Cyganiak	Digital Enterprise Research Institute, NUI Galway, Ireland
Florian Daniel	University of Trento, Italy
Olga De Troyer	Vrije Universiteit Brussel, Belgium
Oscar Díaz	University of the Basque Country, Spain
Damiano Distante	Unitelma Sapienza University, Italy
Peter Dolog	Aalborg University, Denmark
Suzanne Embury	University of Manchester, UK
Flavius Frasincar	Erasmus University Rotterdam, The Netherlands
Piero Fraterno	Politecnico di Milano, Italy
Martin Gaedke	Chemnitz University of Technology, Germany
Dragan Gasevic	Athabasca University, Canada
Athula Ginige	University of Western Sydney, Australia
Jaime Gomez	University of Alicante, Spain
Michael Grossniklaus	Portland State University, USA
Volker Gruhn	University of Leipzig, Germany
Simon Harper	University of Manchester, UK
Andreas Harth	AIFB, Karlsruhe Institute of Technology, Germany
Olaf Hartig	Humboldt-Universität zu Berlin, Germany
Bernhard Haslhofer	University of Vienna, Austria
Martin Hepp	Bundeswehr University Munich, Germany

VIII Organization

Geert-Jan Houben	TU Delft, The Netherlands
Gerti Kappel	Vienna University of Technology, Austria
In-Young Ko	Korea Advanced Institute of Science and Technology, South Korea
Nora Koch	Ludwig Maximilians University of Munich, Germany
Stefan Kühne	University of Leipzig, Germany
Jens Lehmann	University of Leipzig, Germany
Frank Leymann	Institute of Architecture of Application Systems, Germany
Xuemin Lin	University of New South Wales, Australia
Steffen Lohmann	Universidad Carlos III de Madrid (UC3M), Spain
Maristella Matera	Politecnico di Milano, Italy
Santiago Meliá	University of Alicante, Spain
Christos Mettouris	University of Cyprus, Cyprus
Hamid Motahari	HP Labs
Wolfgang Nejdl	L3S and University of Hannover, Germany
Axel-Cyrille Ngonga Ngomo	University of Leipzig, Germany
Luis Olsina	UNLP, Argentina
Satoshi Oyama	Hokkaido University, Japan
George Pallis	University of Cyprus, Cyprus
George Papadopoulos	University of Cyprus, Cyprus
Oscar Pastor Lopez	Universidad Politécnica de Valencia, Spain
Cesare Pautasso	University of Lugano, Switzerland
Vicente Pelechano	Universidad Politécnica de Valencia, Spain
Alfonso Pierantonio	University of L'Aquila, Italy
Matthias Quasthoff	Hasso-Plattner-Institut, Germany
I.V. Ramakrishnan	SUNY Stony Brook, USA
Gustavo Rossi	UNLP, Argentina
Fernando Sanchez-Figueroa	Universidad de Extremadura, Spain
Daniel Schwabe	PUC-Rio, Brazil
Juan F. Sequeda	The University of Texas at Austin, USA
Michael Sheng	University of Adelaide, Australia
Weisong Shi	Wayne State University, USA
Takehiro Tokuda	Tokyo Institute of Technology, Japan
Riccardo Torlone	Roma Tre University, Italy
Jean Vanderdonckt	Université catholique de Louvain, Belgium
Denny Vrandecic	KIT, Germany
Erik Wilde	UC Berkeley, USA
Marco Winckler	LIIHS-IRIT, Paul Sabatier University, France
Bin Xu	DCST, Tsinghua University, China

Additional Reviewers

Ahmed, Faisal	Mao, Huajian
Baez, Marcos	Mayrhofer, Dieter
Bala, Harish	Mirylenka, Daniil
Barla, Michal	Nguyen, Tung
Berger, Thorsten	Noor, Talal
Bonetta, Daniele	Panach Navarrete, Jose Ignacio
Book, Matthias	Peternier, Achille
Borodin, Yevgen	Pröll, Birgit
Brosch, Petra	Reiter, Michael
Brückmann, Tobias	Rezgui, Abdelmounaam
Conejero, Jose Maria	Schleicher, Daniel
Ermilov, Timofey	Soi, Stefano
Fernandez, Adrian	Sonntag, Mirko
Gonzalez-Huerta, Javier	Sun, Kewu
Hermida, Jesus M.	Tramp, Sebastian
Kramár, Tomáš	Tvarožek, Mihal
L, Sen	Wang, Lijuan
Lage, Ricardo	Wetzstein, Branimir
Langer, Philip	Wimmer, Manuel
Lew, Philip	Yao, Lina
Liegl, Philipp	Ye, Penjie
Linaje, Marino	Yesilada, Yeliz
Lins De Vasconcelos, Alexandre	Zaveri, Amapali
Marcos	Zhan, Liming
Luo, Sen	Zhao, Xiang

Table of Contents

Invited Papers

The Anatomy of a Multi-domain Search Infrastructure	1
<i>Stefano Ceri, Alessandro Bozzon, and Marco Brambilla</i>	
How the Minotaur Turned into Ariadne: Ontologies in Web Data Extraction	13
<i>Tim Furche, Georg Gottlob, Xiaonan Guo, Christian Schallhart, Andrew Sellers, and Cheng Wang</i>	

Research Track Papers

Analyzing Cross-System User Modeling on the Social Web	28
<i>Fabian Abel, Samur Araújo, Qi Gao, and Geert-Jan Houben</i>	
Parallel Data Access for Multiway Rank Joins	44
<i>Adnan Abid and Marco Tagliasacchi</i>	
Assessing Fault Occurrence Likelihood for Service-Oriented Systems	59
<i>Amal Alhosban, Khayyam Hashmi, Zaki Malik, and Brahim Medjahed</i>	
A Strategy for Efficient Crawling of Rich Internet Applications	74
<i>Kamara Benjamin, Gregor von Bochmann, Mustafa Emre Dincturk, Guy-Vincent Jourdan, and Iosif Viorel Onut</i>	
Graph-Based Search over Web Application Model Repositories	90
<i>Bojana Bislimovska, Alessandro Bozzon, Marco Brambilla, and Piero Fraternali</i>	
AdapForms: A Framework for Creating and Validating Adaptive Forms	105
<i>Morten Bohøj, Niels Olof Bouvin, and Henrik Gammelmark</i>	
Design and Implementation of Linked Data Applications Using SHDM and Synth	121
<i>Mauricio Henrique de Souza Bomfim and Daniel Schwabe</i>	
A Quality Model for Mashups	137
<i>Cinzia Cappiello, Florian Daniel, Agnes Koschmider, Maristella Matera, and Matteo Picozzi</i>	
DashMash: A Mashup Environment for End User Development	152
<i>Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Gabriele Sprega, Donato Barbagallo, and Chiara Francalanci</i>	

Learning Semantic Relationships between Entities in Twitter	167
<i>Ilknur Celik, Fabian Abel, and Geert-Jan Houben</i>	
Mobile Mashup Generator System for Cooperative Applications of Different Mobile Devices	182
<i>Prach Chaisatien, Korawit Prutsachainimmit, and Takehiro Tokuda</i>	
A Framework for Concern-Sensitive, Client-Side Adaptation	198
<i>Sergio Firmenich, Marco Winckler, Gustavo Rossi, and Silvia Gordillo</i>	
Instantiating Web Quality Models in a Purposeful Way	214
<i>Philip Lew and Luis Olsina</i>	
Reusing Web Application User-Interface Controls	228
<i>Josip Maras, Maja Štula, and Jan Carlson</i>	
Tools and Architectural Support for Crowdsourced Adaptation of Web Interfaces	243
<i>Michael Nebeling and Moira C. Norrie</i>	
A Layered Approach to Revisitation Prediction	258
<i>George Papadakis, Ricardo Kawase, Eelco Herder, and Claudia Niederée</i>	
Improving the Exploration of Tag Spaces Using Automated Tag Clustering	274
<i>Joni Radelaar, Aart-Jan Boor, Damir Vandic, Jan-Willem van Dam, Frederik Hogenboom, and Flavius Frasincar</i>	
A Semantic Web Annotation Tool for a Web-Based Audio Sequencer	289
<i>Luca Restagno, Vincent Akkermans, Giuseppe Rizzo, and Antonio Servetti</i>	
CloudFuice: A Flexible Cloud-Based Data Integration System	304
<i>Andreas Thor and Erhard Rahm</i>	
Bootstrapping Trust of Web Services through Behavior Observation	319
<i>Hamdi Yahyaoui and Sami Zhioua</i>	
Parallel Distributed Rendering of HTML5 Canvas Elements	331
<i>Shohei Yokoyama and Hiroshi Ishikawa</i>	
Formal Modeling of RESTful Systems Using Finite-State Machines	346
<i>Ivan Zuzak, Ivan Budiselic, and Goran Delac</i>	

Poster and Demo Papers

Knowledge Spaces	361
<i>Marcos Baez, Fabio Casati, and Maurizio Marchese</i>	
Exploratory Multi-domain Search on Web Data Sources with Liquid Queries	363
<i>Davide Francesco Barbieri, Alessandro Bozzon, Marco Brambilla, Stefano Ceri, Chiara Pasini, Luca Tettamanti, Salvatore Vadacca, Riccardo Volonterio, and Srdan Zagorac</i>	
Model-Based Dynamic and Adaptive Visualization for Multi-domain Search Results	367
<i>Alessandro Bozzon, Marco Brambilla, Luca Cioria, Piero Fraternali, and Maristella Matera</i>	
A Constraint Programming Approach to Automatic Layout Definition for Search Results	371
<i>Alessandro Bozzon, Marco Brambilla, Laura Cigardi, and Sara Comai</i>	
Adaptive Mobile Web Applications: A Quantitative Evaluation Approach	375
<i>Heiko Desruelle, Dieter Blomme, and Frank Gielen</i>	
A Personality Mining System for Automated Applicant Ranking in Online Recruitment Systems	379
<i>Evanthia Faliagka, Lefteris Kozanidis, Sofia Stamou, Athanasios Tsakalidis, and Giannis Tzimas</i>	
Development of the Evaluation Form for Expert Inspections of Web Portals	383
<i>Andrina Granić, Ivica Mitrović, and Nikola Marangunić</i>	
WebSoDa: A Tailored Data Binding Framework for Web Programmers Leveraging the WebSocket Protocol and HTML5 Microdata	387
<i>Matthias Heinrich and Martin Gaedke</i>	
Towards User-Centric Cross-Site Personalisation	391
<i>Kevin Koidl, Owen Conlan, and Vincent Wade</i>	
Tool Support for a Hybrid Development Methodology of Service-Based Interactive Applications	395
<i>Christian Liebing, Marius Feldmann, Jan Mosig, Philipp Katz, and Alexander Schill</i>	
A Comparative Evaluation of JavaScript Execution Behavior	399
<i>Jan Kasper Martinsen, Håkan Grahn, and Anders Isberg</i>	

XIV Table of Contents

Designing a Step-by-Step User Interface for Finding Provenance Information over Linked Data	403
<i>Enayat Rajabi and Mohsen Kahani</i>	
Towards Behaviorally Enriched Semantic RESTful Interfaces Using OWL2	407
<i>Irum Rauf and Ivan Porres</i>	
Taxonomy for Rich-User-Interface Components: Towards a Systematic Development of RIAs	411
<i>Rosa Romero Gómez, David Díez Cebollero, Susana Montero Moreno, Paloma Díaz Pérez, and Ignacio Aedo Cuevas</i>	
NAVTAG - A Network-Theoretic Framework to Assess and Improve the Navigability of Tagging Systems	415
<i>Christoph Trattner</i>	
Author Index	419

The Anatomy of a Multi-domain Search Infrastructure

Stefano Ceri, Alessandro Bozzon, and Marco Brambilla

Dipartimento di Elettronica e Informazione,
Politecnico di Milano, P.zza Leonardo Da Vinci 32,
20133 Milan, Italy
{name.surname}@polimi.it

Abstract. Current search engines do not support queries that require a complex combination of information. Problems such as “Which theatre offers an at least-three-stars action movie in London close to a good Italian restaurant” can only be solved by asking multiple queries, possibly to different search engines, and then manually combining results, thereby performing “data integration in the brain.” While searching the Web is the preferred method for accessing information in everyday’s practice, users expect that search systems will soon be capable of mastering complex queries. However, combining information requires a drastic change of perspective: a new generation of search computing systems is needed, capable of going beyond the capabilities of current search engines. In this paper we show how search computing should open to modular composition, as many other kinds of software computations. We first motivate our work by describing our vision, and then describe how the challenges of multi-domain search are addressed by a prototype framework, whose internal “anatomy” is disclosed.

Keywords: Web information retrieval, multi-domain query, search computing, software architecture, modular decomposition.

1 Introduction

Search is the preferred method to access information in today’s computing systems. The Web, accessed through search engines, is universally recognized as the source for answering users’ information needs. However, offering a link to a Web page does not cover all information needs. Even simple problems, such as “Which theatre offers an at least-three-stars action movie in London close to a good Italian restaurant”, can only be solved by searching the Web multiple times, e.g. by extracting a list of the recent action movies filtered by ranking, then looking for movie theatres, then looking for Italian restaurants close to them. While search engines hint to useful information, the user’s brain is the fundamental platform for information integration.

An important trend is the availability of new, specialized data sources – the so-called “long tail” of the hidden Web of data. Such carefully collected and curated data sources can be much more valuable than information currently available in Web pages; however, many sources remain hidden or insulated, in the lack of software technologies for bringing them to surface and making them usable in the search context. We believe that in the future a new class of search computing systems will support the publishing and integration of data sources; the user will be able to select

sources based on individual or collective trust, and systems will be able to route queries to such sources and to provide easy-to-use interfaces for combining them within search strategies, at the same time rewarding the data source owners for each contribution to effective search. Efforts such as Google's Fusion Tables show that the technology for bringing hidden data sources to surface is feasible. We argue then that a new economical model should promote data sharing, giving incentives to owners - for data publishing - and to brokers - for building new search applications by composing them.

The Search Computing project (SeCo) [11][12] aims at building concepts, algorithms, tools and technologies to support complex Web queries, through a new paradigm based on combining data extraction from distinct sources and data integration by means of specialized integration engines. The project has the ambitious goal of lowering the technological barrier required for building complex search applications, thereby enabling the development of many new applications.

In essence, the new requirements imposed to search call for a component-oriented view of search computations. While the current landscape in Web search is dominated by giant companies and monolithic systems, we believe that a new generation of search computing systems needs to be developed, with a much more composite software organization, addressing the needs of a fragmented market. Generic search systems are already dominated by domain-specific vertical search systems, e.g. with travels and electronic bookstores. When the threshold complexity of building such verticals will be lowered, a variety of new market sectors will become more profitable. In several scenarios, search-enabled Web access will grow in interest and value when SMEs or local businesses will see the opportunity of building search applications tailored to their sale region, which integrate "local" and "global" information.

Therefore, building a search computing systems requires coping with several issues. On one side, new methods and algorithms need to be embedded into software modules devised to solve the diverse sub-problems, including query specification, query planning and optimization, query execution, service invocation, user interface rendering, and so on. On the other side, these software modules must be integrated within a software environment supporting collaborative executions. This paper describes first our vision on how search systems should evolve towards modular software components and interfaces; then, as a guarantee of the feasibility of such vision, we describe the "anatomy" of the Search Computing framework that we are currently developing, discussing the main architectural challenges and solutions.

2 Vision

Supporting search over data sources requires a new class of data services, denoted as **Search Services**, and of data integration systems, denoted as **Search Computing Systems**.

2.1 Data Provisioning

An increasing number of data sets is becoming available on the Web as (semi) structured data instead of user-consumable pages. Linked Data plays a central role in this, thanks to initiatives such as W3C Linked Open Data (LOD) community project,

which are fostering LD best practice adoption [2]. An important aspect of LD is their use of universal references for data linking; this aspect raises the hopes of solving the data-matching problem, which has so far limited the practical applicability of data integration methods.

LD and other open or proprietary data are made available through Web APIs (e.g., see Google Places API) and/or search-specific languages (e.g., see the Yahoo Query Language (YQL) framework [19]). Methods, models and tools are being devised for efficiently designing and deploying applications upon such new data services and data access APIs. However, data access and linking so far has not been concerned with data search. The quality of these data sources can be fully exploited with the growth of new search applications, which federate and compose data sources.

Data services must expose ranking criteria and must be optimized for ranked retrieval; top-k queries are well supported by relational sources, while extensions in this direction are being pursued by the Semantic Web community, e.g. through language extensions covering tuple orderings in time and rank (e.g. Continuous-Sparql and RankSparql).

2.2 Service Composition

The efficient support of such data services requires mastering both data and control dependencies, and strategy optimization must consider rank aggregation, optimal result paging, and so on [7]. When data sources must be joined, the join operation must take into account ranking; join can either be based on exact methods, according to the rank-join theory, or on approximate methods, that favor the speed of result production [15]. Data integration strategies should aim at obtaining a suitable number of results (e.g., neither too few, nor too many). Normally, a computation should not be set up so as to exhaust a searchable data source, as the user is rarely interested to inspect all of them. The ultimate controller is the user, who sees service results or their compositions and can halt their production.

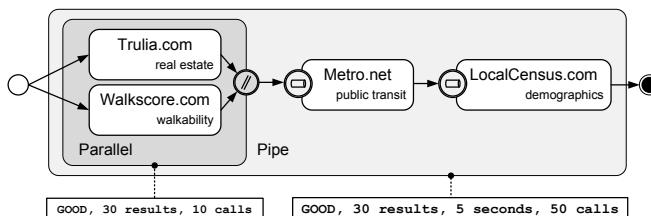


Fig. 1. Search Service integration example, high level view

To be concrete, think about developing a real estate search system in a given region, by using both global and local resources. The high-level process orchestration offered by such system may consist in the composition of data sources providing real estate offers (e.g., Trulia.com or Zillow.com) with local services (e.g., Metro.com for public transport) and indexes (e.g., WalkScore.com or LocalCensus.com), as shown in Figure 1. In the example, the Trulia and WalkScore services are invoked in parallel to extract housing offers and the walkability index (i.e., a value describing if the

neighborhood is walking-friendly, in terms of services, shops, and restaurants at walking distance) of the various districts. This parallel execution is set to extract at most 30 good results by using at most 10 service calls. Then, results are piped to the Metro and LocalCensus services, which extract demographics and crime rate information for each solution. Three global thresholds are set on the number of results, on the time limit and on the number of service calls (30 results, 5 seconds, 50 calls respectively). Thresholds correspond to execution constraints: execution halts if any threshold is met. Notice that this is not a simple mashup of service, as it includes, e.g., complex optimization strategies and result composition logics [6].

From a system-oriented point of view, search-oriented engines for data integration should be deployed in “cloud computing” environments, to enable flexibility and scalability for application developers. In this way, the complexity of engine design and optimization would be totally outsourced and become transparent to the developer. In essence, the developer should not only resort to external data sources, but also to external data integration systems.

2.3 User interaction

From the end user standpoint, the way in which the informative need can be described may be crucial for determining the acceptance of a search application. Hence, different query formulations and user interactions should be allowed. In particular, one can think of three types of query: one-shot form-based queries, natural language queries, and exploratory queries.

One-shot queries based on forms correspond to the behavior of several domain-specific search engines in which the objective of the search is predefined and the user submits a fixed set of parameters for finding the information. Typical examples are flight search or hotel search applications. In case of multi-domain queries one can expect a more complex form, comprising more fields and parameters. In this case the set of focal entities are typically known *a priori* and the input parameters are exactly matching the exposed fields in the form.

Natural language queries offer instead the maximum freedom to the user, who can express his information needs in a free text form. The submitted sentence might contain query parameters, references to the objects of interest, and possibly information on the expected results (i.e., the expected output of the search). NL queries represent an example of incremental query methods, where incrementality is achieved with a process by which, given his/her information need, the user is led to full specification of the most suitable search services to satisfy it. This scenario is much harder to deal with, as not only the text might not specify the complete set of expected input parameters for producing a response, but it also requires a separation of the original query into a number of sub-queries based on probabilistic mappings of concepts and properties.

Finally, **exploratory queries** are queries where, given the current context of interaction, the user is able to follow links to connected concepts, thus adding a new query fragments or rolling back to a previous result combination. Exploratory queries are by nature incremental. To give users some flexibility in exploring the result, we have proposed the Liquid Query paradigm [3], which allows users to interact with the search computing result by asking the system to produce “more result combinations”,

or “more results from a specific service”, or “performing an expansion of the result” by adding a sub-query which was already planned while configuring the query.

By means of such paradigm, the user is supported in expressing fully exploratory queries, starting from an initial status with no predefined query, and enabling a progressive, step-by-step construction of the query itself. The new paradigm consists of exploring a network of connected resources, where each resource corresponds to a clearly identified real-world concept (an “hotel”, a “flight”, a “hospital”, a “doctor”), and the connections have predefined semantics (“hotels” are close to “restaurants”, “doctors” care “diseases” and are located at “hospitals”). Such network, called the “Semantic Resource Framework”, is built as a conceptual description of the search services exploited by the multi-domain framework. The proposed exploration paradigm exploits query expansion and result tracking, giving the user the possibility to dynamically selecting and deselecting the object instances of interest, and move “forward” (adding one node to the query) or “backward” (deleting one node) in the resource graph. Result presentation paradigms support the exploration by visualizing instances of different objects in separate lists, at the same time displaying the combinations they belong to and their global rank. This view allows users to focus at each step on the new results, and therefore is most suitable for a progressive exploration.

2.4 Application Development

A complex search application can be generated from such a high-level design after a preliminary data source registration. During such process, sources become known in terms of: the concepts that they describe (conceptual view), their “access pattern”, i.e. the input-output parameters and supported ranking (logical view), and the actual supported interaction protocol, with a variety of quality parameters (physical view). The registration process should be as simple and encompassing as possible, and include a mix of top-down acts (when starting from concepts) and bottom-up acts (when starting from data source schemas or service calls).

We envision semi-automatic tagging of sources during registration and use, and we suggest that tags be extracted from general ontologies (such as Yago) so as to build an abstract view of the sources that can be helpful in routing queries to them [18]. The search system Kosmix [17] already uses such an approach for describing data sources that is sufficient for selecting relevant sources and driving simple queries to them, but it currently does not support the combination of data sources through operations. Combining data sources within queries requires going beyond the registration or discovery of data sources as entities, and supporting the registration of the relationships semantically relating such entities, at the conceptual, logical, and physical level. Relationship discovery is the most difficult step and the key to ease service registration and application development; early work on linked data and on knowledge extraction (e.g. from social networks) proves that relationship discovery is becoming feasible.

Application developers could act as brokers of new search applications built by assembling data sources, exposed as search services. We envision a growing market of specialized, localized, and sophisticated search applications, addressing the long tail of search needs (e.g., the “gourmet suggestions” about slow-food offers in given geographic regions). In this vision, large communities of service providers and brokers (e.g., like ProgrammableWeb.com and Mashape.com for mashups) could be

empowered by support design environment and tools for executing search service compositions and orchestrations. Thanks to the lowering of programming barriers one could expect an ultimate user's empowerment, whereby end users could compose data sources at will from predefined data source registries and collections; e.g., the orchestration of Figure 1 could be built by starting from simple, menu-driven interfaces where the user is just asked to select the concepts and the orchestration is then inferred. We also envision that, with suitable ontological support and possibly within a narrow domain, queries could be generated from keywords, as with conventional search engines.

3 Reference Architecture

The Search Computing project covers many research directions, which are all required in order to provide an overall solution to complex search. The core of the project is the technology for search service integration, which requires both theoretical investigation and engineering of efficient technological solutions. The core theory concerns the development of result integration methods that not only denote "top-k optimality", but also the need of dealing with proximity, approximation, and uncertainty. A number of further research dimensions complement such core. Service integration requires solving schema integration problems, as well as ontological description and labeling of resources to facilitate queries. Efficient execution requires optimization, caching, and a server configuration supporting scalability through distribution and parallelism. Support of user interaction requires powerful client-side computations, with rich interfaces for assisting users in expressing their needs and exploring the information. Design tools for building Search Computing applications employ mashup-based solutions and specialized visualization widgets.

Therefore, Search Computing systems deal with many typical Web engineering problems. Figure 2 shows the architecture of the Search Computing system. The software modules in Figure 2 are vertically subdivided into processing modules, repositories, and design tools, and vertically organized as a two-tier, three-layer infrastructure, with the client tier dedicated to user interaction and the server tier further divided into a control layer and execution layer; the client-server separation occurs between processing layers and repositories, and communications are performed using Web-enabled channels. Tools address the various phases of interaction.

3.1 Processing Modules

We describe processing modules bottom-up, starting with the *Execution Layer*. The lower module, the *Service Invocation Framework*, is in charge of invoking services that query the data sources. Such services typically have few input parameters (which are used to complete parametric queries) and produce results constituted by a "chunk" of tuples, possibly ranked, each equipped with a tuple-id; thus, a service call maps given input parameters to a given chunk of tuples. The framework includes built-in wrappers for the invocation of several Web based infrastructures (e.g., YQL, GBASE), query end-point (e.g. SPARQL) and resources (e.g., WSDL- and REST-based Web services). It also supports the invocation of legacy and local data sources.

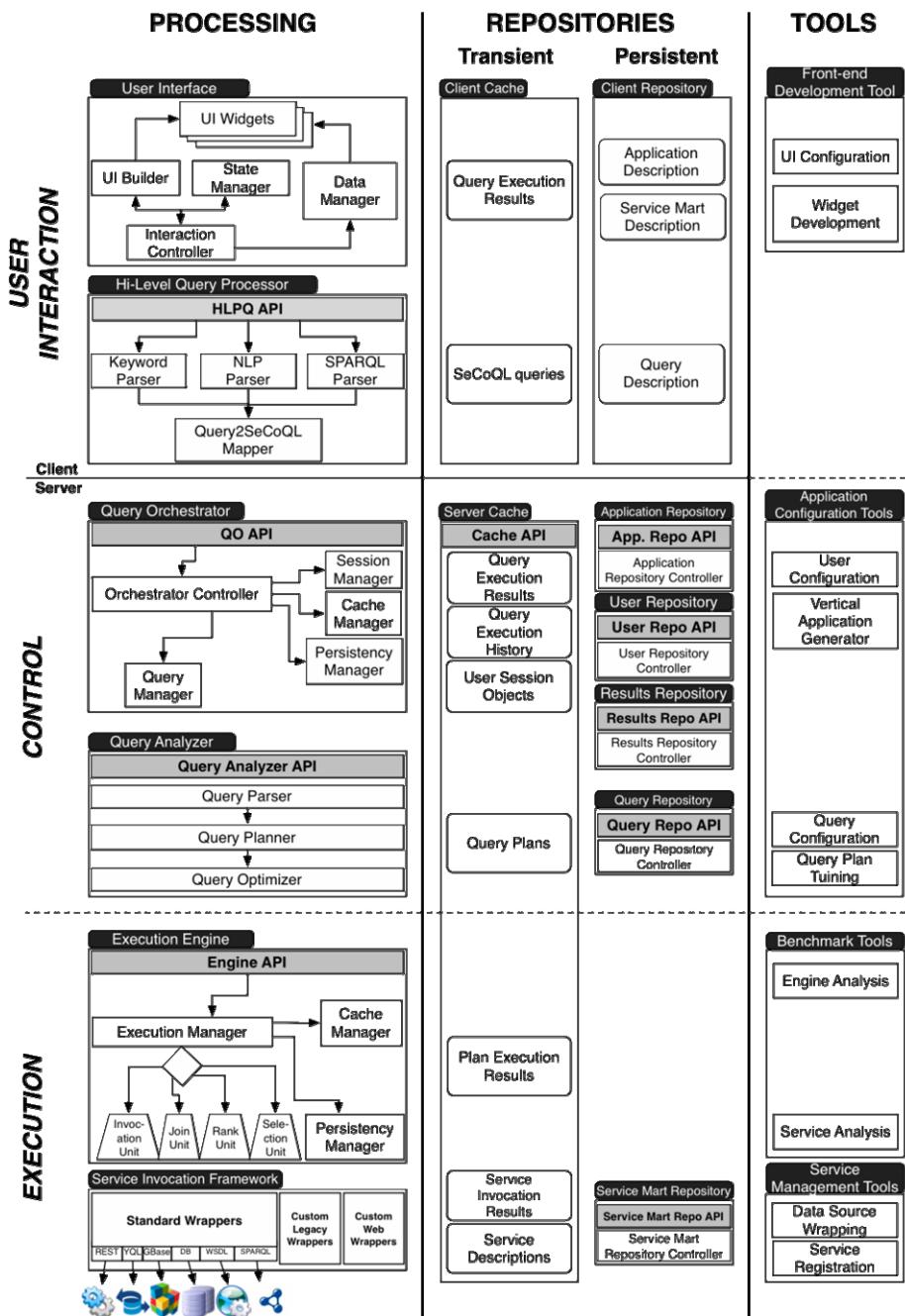


Fig. 2. The architecture of the Search Computing platform

The *Execution Engine* is a data- and control-driven query engine specifically designed to handle multi-domain queries [9]. It takes in input a reference to a query plan and executes it, by driving the invocation of the needed services through the *Service Invocation Framework*. The *Execution Engine* includes a *Cache* and a *Persistency Manager* module, devoted to in-session and cross-session level caching of results.

The *Control Layer*, as the name suggests, is the controller of the architecture; it is designed to handle several system interactions (such as user session management and query planning) and it embodies the *Query Analyzer* and the *Query Orchestrator*. The *Query Analyzer* is a component devoted to the parsing, planning, and optimization of queries (possibly expressed in a declarative language) into query plans to be executed by the *Execution Engine*. The *Query Orchestrator* is the main access point to the platform, as it acts as a proxy toward all the internal components, offering a set of APIs. The *Query Orchestrator* also handles the management of user sessions and authentication, and communicates with the *Execution Engine* to perform queries; each query is univocally identified, as well as the result set produced by its execution.

The *User Interaction Layer* is the front-end of the SeCo system, and it embodies the *User Interface* and the *High-Level Query Processor*. The latter is a component designed to allow users to express unstructured queries (e.g., a set of keywords, a natural language sentence, or a structured English sentence) or declarative queries expressed over an ontology (or, more in general, a schema), which are translated into processable multi-domain queries. The *User Interface*, instead, is a client-side Rich Internet Application dynamically configured according to a designed SeCo application. It accepts in input user commands and produces queries and query expansion commands for the *Query Orchestrator*.

The rationale of this architecture is a clear separation of modules by functions. Thus, the lower components separate the service interaction logics from the service composition logics, the separation between the orchestrator and the engine guarantees that the former is focused on session management while the latter is focused on efficient execution of one-shot queries, thereby simplifying the engine – which is the most critical component. The design of the *Query Planner* from a component external to the query engine allows its invocation from the *Query Orchestrator*, thus offering planning services to several other components. Finally, the separation of the client-side query processor from the user interface guarantees that all the specific processing depending on the type of input is performed in the former, while all the device-specific aspects of presentation are considered in the latter.

3.2 Repositories

The *Repository* contains the set of components and data storages used by the system to persist the artifacts required for its functioning. On the server side, the *Service Mart Repository* contains the description of the search services consumed by the system, represented at different level of abstraction; it is accessed by the *Query Orchestrator* to provide users with information for navigating the service resource graph while it provides to the *Execution Engine* the access to services. The *Query Repository* and *Results Repository* are used in order to persist query definitions and query results which are heavily used, while the *User Repository* and *Application Repository* store, respectively, a description of users accessing an application and of the configuration

files (query configuration, result configuration, etc.) required by the user interface for an application.

On the client side, three persistent repositories store have been designed, respectively: the *Query Descriptions*, which are used by the High-level Query Processor as reference definitions of the query models; the *Service Mart Descriptions* and the *Application Descriptions*, managed by the user interface on the user's browser to persistently cache application's configuration files and service descriptions, thus reducing the time required for the application to be downloaded and started.

3.3 Caching

The efficient handling of query executions is guaranteed by the combined design of the processing module and of a shared and distributed caching system, which maximizes artifacts reuse while minimizing redundant computation and network data transfer: for instance, caching the results of service invocations allows for faster query answers (as the execution time of a query is typically dominated by the response latency of the search service); repeated executions of the same query or subquery allows for reduced computations, thus enhancing the scalability of the system. Client-side caching of query results minimize the need for client-server round-trip, thus improving the user experience.

Figure 3 illustrates an extract of the sequence diagram of the interactions occurring within the processor layer and its cache system: chunks extracted by each service calls are cached; query plans are cached with the data which are produced by its execution; query execution histories are cached together with user inputs and results at each step. Caches indexes support a fast lookup, e.g. query results are accessed by query plan, user input and chunk number; cache items remain valid throughout a user session (to guarantee consistency in the retrieved data, while improving the reactivity of the system) or across several sessions (to maximize reuse, when possible), unless explicitly invalidated. Query execution occurs by systematically checking whether a resource exists in the cache, else it is produced: when a query created by the user interface, the *Query Orchestrator*, which manage user session histories, looks up in the cache for existing plans associated with such query; if not hit is found, the *Query Orchestrator* interacts with the *Query Planner* to create a new query plan, and looks again in the cache for existing results associated with the plan invocation. If no results have been cached, the control is redirected to the *Execution Engine*, which, in turn, progressively invokes the services included in the query plan to retrieve (possibly cached) data and compose results. A service invocation in a query plan can be directed to another query plan; therefore, the execution engine is able to recursively execute plans, re-using the results of previous executions when possible. The *Query Orchestrator* also handles user session objects, to manage the navigation *history* of the user during exploratory search tasks: the state of navigation is made of a set of user interactions on the system, together with the produced results.

3.4 Tools

To support and configure the set of objects and components involved in the SeCo architecture, a tool suite has been devised that comprises a variety of instruments. The

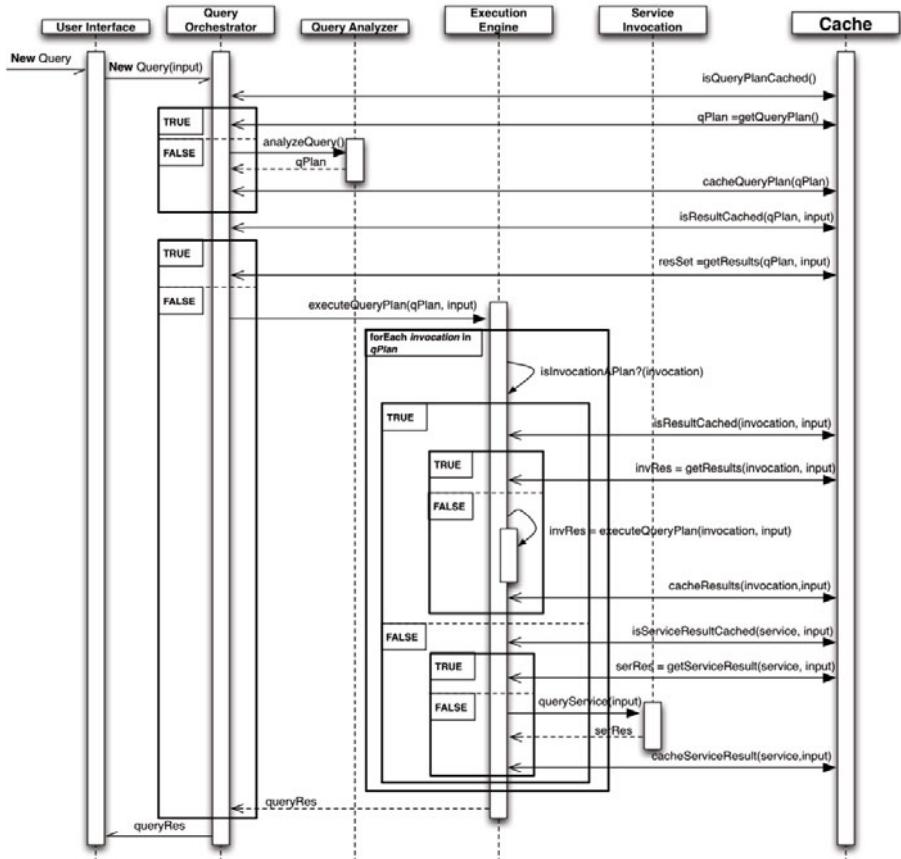


Fig. 3. Sequence diagram showing processing and cache modules

tool suite has been structured as an online development platform in which developers can login and, according to their role, access the right set of tools for building SeCo applications. The availability of the tools as online applications aims at increasing SeCo application design productivity, by reducing the time to deployment and avoiding the burden of downloading and installing software.

At the service management level, tools are crucial for service registration and wrapping. The *Service wrapping tool* allows normalization of service interfaces by supporting the definition of service wrappers, i.e., software components that make heterogeneous services fit into the search computing search services conventions. The *registration tool* [10] consists of a set of facilities for the mapping of concrete services to their conceptual descriptions in terms of service marts, access patterns, and connection patterns. We are currently extending the service registration process so that terms used in naming concepts are extracted from Yago [17], a general-purpose ontology. The tools consist of mapping-based interfaces that allow picking elements from the service input/outputs (and domain descriptions) and populating the

conceptual service mart models. Furthermore, a *service analysis tool* supports monitoring and evaluation of the service performance. At the engine level, the *execution analysis* tool is a benchmarking dashboard that allows monitoring and fine-tuning of the engine behavior. The execution of queries can be visually monitored, recorded, and compared with other executions (e.g., for evaluating the best execution strategies of the same query, or for assessing the bottlenecks of an execution) [5].

The *application configuration tools* allow designers to define applications based on the composition of search services: the *query configuration tool* supports the visual definition of queries as subsets of the service description model with predefined condition templates. The tool supports the designer in exploring the service repository, through visual navigation, selecting the services of interest for the application and the respective connection patterns, and defining the conditions upon them. The *query plan tuning tool* is visual modeling environment that allows SeCo experts to refine the edit query plans specified according to the Panta Rhei notation. The *user configuration tool* allows one to define user profile properties and the *vertical application generator* produces application models that are stored in the application repository.

The client side issues are addressed by two tools: the *UI configuration tool* aims at supporting the design of the interface of the query submission form and of the result set, together with the default settings for the application and the allowed Liquid Query operations [3]. The *widget development environment* consists of a framework in which the developer can encode his own new visual components to be inserted into the pages (e.g., new result visualizers or new query submission widgets).

4 Conclusions

This paper presented our vision for a novel class of search systems, advocating that a new generation of search infrastructures with a modular software organization is required for addressing the needs of a fragmented market of new search applications. We also showed how this vision is partially instantiated by the prototype architecture currently under development within the Search Computing project. Demos providing some evidence of the feasibility of this approach are presented at WWW [4], ACM-Sigmod [5] and ICWE [1].

Acknowledgements. This research is part of the Search Computing (SeCo) project, funded by ERC, under the 2008 Call for "IDEAS Advanced Grants" (<http://www.search-computing.org>). We wish to thank all the contributors to the project.

References

- [1] Barbieri, D., Bozzon, A., Brambilla, M., Ceri, S., Pasini, C., Tettamanti, L., Vadacca, S., Volonterio, R., Zagorac, S.: Exploratory Multi-domain Search on Web Data Sources with Liquid Queries. In: ICWE 2011 Conference, Demo session, Paphos, Cyprus (June 2011)
- [2] Bizer, C., Heath, T., Idehen, K., Berners-Lee, T.: Linked Data on the Web. In: Proceedings WWW 2008, Beijing, China (2008)

- [3] Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P.: Liquid Query: Multi-Domain Exploratory Search on the Web. In: WWW 2010, Raleigh, NC, pp. 161–170. ACM, New York (2010)
- [4] Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P., Vadacca, S.: Exploratory search in multi-domain information spaces with Liquid Query. In: WWW 2011 Conference, Demo session, (March 31, 2011)
- [5] Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P., Vadacca, S.: Exploratory search in multi-domain information spaces with Liquid Query. In: Bozzon, A., Brambilla, M., Ceri, S., Corcoglioniti, F., Fraternali, P., Vadacca, S. (eds.) Search Computing: Multi-domain Search on Ranked Data, ACM-Sigmod 2011 Conference, Demo session (June 2011)
- [6] Braga, D., Campi, A., Ceri, S., Raffio, A.: Joining the results of heterogeneous search engines. *Inf. Syst.* 33(7-8), 658–680 (2008)
- [7] Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of Multi-Domain Queries on the Web. In: VLDB 2008, Auckland, NZ (2008)
- [8] Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Mashing Up Search Services. *IEEE Internet Computing* 12(5), 16–23 (2008)
- [9] Braga, D., Grossniklaus, M., Corcoglioniti, F., Vadacca, S.: Efficient Computation of Search Computing Queries. In: Ceri, S., Brambilla, M. (eds.) Search Computing II. LNCS, vol. 6585, pp. 141–155. Springer, Heidelberg (2011)
- [10] Brambilla, M., Tettamanti, L.: Tools Supporting Search Computing Application Development. In: Ceri, S., Brambilla, M. (eds.) Search Computing II. LNCS, vol. 6585, pp. 169–181. Springer, Heidelberg (2011)
- [11] Ceri, S., Brambilla, M. (eds.): Search Computing II. LNCS, vol. 6585. Springer, Heidelberg (March 2011)
- [12] Ceri, S., Brambilla, M. (eds.): Search Computing. LNCS Book, vol. 5950. Springer, Heidelberg (March 2010) ISBN 978-3-642-12309-2
- [13] Danescu-Niculescu-Mizil, C., Broder, A.Z., Gabrilovich, E., Josifovski, V., Pang, B.: Competing for users' attention: on the interplay between organic and sponsored search results. In: WWW 2010, Raleigh, NC, pp. 291–300. ACM, USA (2010)
- [14] Google: Fusion Tables (2009), <http://tables.googlelabs.com/>
- [15] Ilyas, I., Beskales, G., Soliman, M.: A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.* 40(4) (2008)
- [16] Parameswaran, A., Das Sarma, A., Polyzotis, N., Widom, J., GarciaMolina, H.: Human-Assisted Graph Search: It's Okay to Ask Questions. In: PVLDB, vol. 4(5), pp. 267–278 (February 2011)
- [17] Rajaraman, A.: Kosmix: High Performance Topic Exploration using the Deep Web. In: VLDB 2009, Lyon, France (2009)
- [18] Suchanek, F., Bozzon, A., Della Valle, E., Campi, A.: Towards an Ontological Representation of Services in Search Computing. In: Ceri, S., Brambilla, M. (eds.) Search Computing II. LNCS, vol. 6585, pp. 101–112. Springer, Heidelberg (2011)
- [19] YQL (2009), <http://developer.yahoo.com/yql/>

How the Minotaur Turned into Ariadne: Ontologies in Web Data Extraction*

Tim Furche, Georg Gottlob, Xiaonan Guo, Christian Schallhart,
Andrew Sellers, and Cheng Wang

Department of Computer Science, University of Oxford, UK
`{firstname.lastname}@cs.ox.ac.uk`

Abstract. Humans require automated support to profit from the wealth of data nowadays available on the web. To that end, the linked open data initiative and others have been asking data providers to publish structured, semantically annotated data. Small data providers, such as most UK real-estate agencies, however, are overburdened with this task—often just starting to move from simple, table- or list-like directories to web applications with rich interfaces.

We argue that fully automated extraction of structured data can help resolve this dilemma. Ironically, automated data extraction has seen a recent revival thanks to ontologies and linked open data to guide data extraction. First results from the DIADEM project illustrate that high quality, fully automated data extraction at a web scale is possible, if we combine domain ontologies with a phenomenology describing the representation of domain concepts. We briefly summarise the DIADEM project and discuss a few preliminary results.

1 Introduction

The web has changed how we do business, search for information, or entertain ourselves to such a degree that saying so has become a platitude. The price for that success is that every business must maintain a website to stay operational. For example, even the smallest real estate agency¹ needs a searchable website and must spend considerable effort to be both found on Google and integrated into the major real estate aggregators. Businesses have reluctantly accepted this cost for doing business as a price for higher visibility—reluctantly, as aggregators present long list of normalised results from all agencies. Thus, agencies have become dependent on dominant aggregators where it is hard to distinguish oneself by reputation, service, or novel features rather than price. Google, on the other hand, is able to pick up on reputation of agencies to some extent, but does very

* The research leading to these results has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 246858 (DIADEM).

¹ Of which there are over ten thousand in the UK alone, some offering just a handful of properties, e.g., focused on a single burrow of Oxford.

poorly on property searches. This observation holds nowadays for many product domains, with price comparison systems becoming dominant.

One of the goals of the semantic web, linked open data and similar initiatives is to address this dependency on few, dominant aggregators: Let real estate agencies publish their data in a structured format such as RDF with an appropriate ontology describing the schema of that data. Then, search engines can be adapted for object search, i.e., finding the property best fitting specific criteria (such as price range and location). In contrast to aggregators, object search engines should be able to pick up any properly structured data published on the web, just as search engines pick up any properly published HTML page. To publish structured data, an agency has to find a suitable business vocabulary (possibly extending it for novel features of its properties) and mark up its properties according to that schema.

For most agencies, the burden of maintaining and promoting a traditional, HTML web page already incurs considerable cost, requiring expensive training and specialised personnel. Publishing high quality structured data using the most relevant ontologies is a task well beyond the expertise of most agencies: First, there are still few good standardised domain ontologies. In the real estate domain, previous attempts at standardising the vocabulary (e.g., as XML schemata for interchange and use in aggregators) have largely failed, partially as properties are not quite commodities yet. Second, small agencies often manage their properties in an ad-hoc fashion and have little or no knowledge (or need) for proper information management. These concerns are reflected in the fact that the growth of linked open data is certainly driven by adoption in governments, large nonprofits, and major companies.

The availability of structured data from all these agencies promises a fairer, more competitive property market—but with considerable sacrifice. We are stuck in a labyrinth of vocabularies, semantic technologies and standards. Every business will be forced to spend considerable resources on publishing proper structured data, even more than they already do for maintaining their web sites. What begins as a call for everyone to participate, may further harm the competitiveness of small businesses.

We argue that the very same reason that makes the labyrinth scary—the ontologies for annotating structured data—can also direct us out of the labyrinth. What is needed is a “**red thread**” program that automatically finds a path for each given website to relevant domain objects. With such a program, we can analyse the pages that people and businesses are publishing, rather than everyone annotating their objects (and solving the related problems again and again). We consider how to turn such pages into structured data along the following questions:

1. How are these objects represented in web sites (*object phenomenology*)?
2. How do humans find these objects (*search phenomenology*)?
3. How to turn that information into a “*red thread*” program.
4. How to extract *all* the data from relevant pages *at scale*.

In the DIADEM project we are working on answering these questions based on a fundamental observation: If we combine domain with phenomenological (object and search) knowledge in an ontology of a domain's web objects, we can automatically derive an extraction program for nearly any web page in the domain. The resulting program produces high precision data, as we use domain knowledge to improve recognition and alignment and to verify the extraction program based on ontological constraints.

DIADEM's "red thread" program is a large set of analysis rules combined with low-level annotators providing a logical representation of a webpage's DOM, including the visual rendering and of textual annotations (based on GATE). Section 2 gives a brief overview of the DIADEM system and its components, including the ontologies and phenomenologies used for web form (Section 3) and object analysis (Section 4).

We generate an extraction program describing the paths through a web site (and individual pages) to get to the actual data. These programs are formulated in OXPath (Section 5), a careful extension of XPath for simulating user interactions with a web site.

More details on DIADEM are available at diadem-project.info.

2 Overview DIADEM Prototype

Figure 1 gives a simplified overview on DIADEM prototype architecture. Every web page is processed in a single sequential pipeline. First we extract the page model from a live rendering of the web page. This model represents logically the DOM, information on the visual rendering, and textual annotations. The textual annotations are generated partially by *domain-specific* gazetteers and rules, but otherwise this part is domain-independent. In the next step, we do an initial classification of web blocks, such as navigation links, advertisements etc. to separate general structures from domain-specific structures and to provide additional clues to object and form analysis. In the third step we use the AMBER prototype, discussed in Section 4 to identify and classify any objects of the domain that may occur on the page. This is done before the form analysis in stage four (using the OPAL prototype from Section 3), as we use the information from the object analysis together with the block classification to decide if navigation links on the page may lead to further data. If the form analysis can identify a form belonging to the target domain, we proceed to fill that form (possibly multiple times). Finally, we extract and crawl links for further exploration of the web site.

Once a site is fully explored, all collected models are passed to the OXPath generator that uses simple heuristics to create a generalised OXPath expression that to be executed with the OXPath prototype for large scale extraction (see Section 5).

This analysis only needs to be repeated if the analyzed site has changed significantly, otherwise the resulting OXPath expression can be used for repeated extraction from the web site.

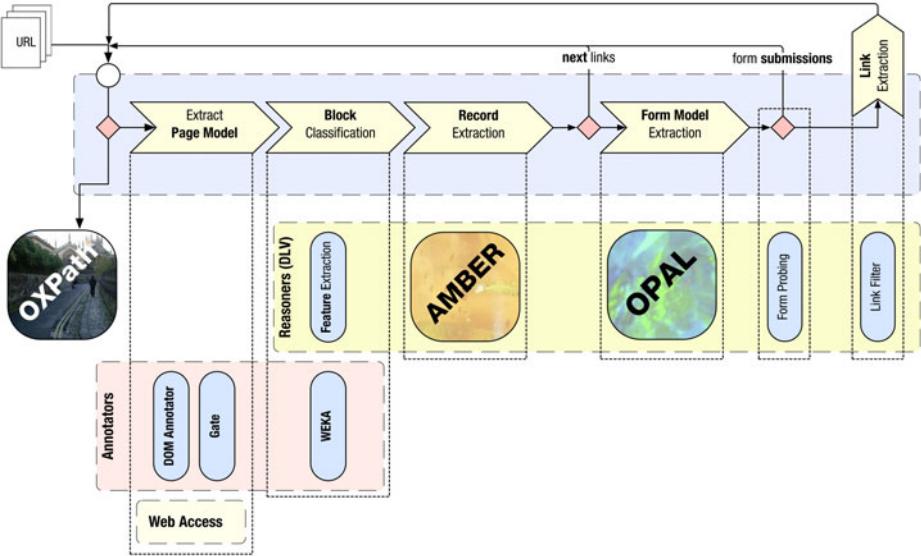


Fig. 1. Overview of the DIADEM 0.1 System

3 Ontologies for Form Analysis

Form understanding, as the gate-way process in automated web data extraction, has been addressed in the context of deep web search [1,2,3,4], web querying [5,6], and web extraction [7]. These approaches focus on observing commonalities of general web forms and exploiting the arising patterns in specifically tailored algorithms and heuristics. However, trying to define a general approach capable of producing high precision results in all domains is not an easy task. Furthermore, by generalizing the assumptions made about web forms, these approaches cannot exploit domain-specific patterns.

To overcome these limitations, we designed an ontology-assisted approach, OPAL (Ontology-based web Pattern Analysis with Logic), for domain-aware form understanding. OPAL analyzes and manipulates on form elements using both general assumptions and domain ontological knowledge. The former adopts several heuristics to provide segmentation and labeling for a form. Form elements and segments are then annotated, classified, and verified using the domain ontology. The link between general form understanding and logical form representation is referred to as the phenomenology, which describes how ontological concepts appear on the web. We have implemented a prototype system for UK real-estate domain, and conducted extensive evaluation of the system on a sampled domain dataset.

OPAL represents information at three successive levels connected by two mappings. Firstly, the page model represents a rendered DOM of a web page through an embedded browser, enriched with visual properties, e.g., bounding boxes of

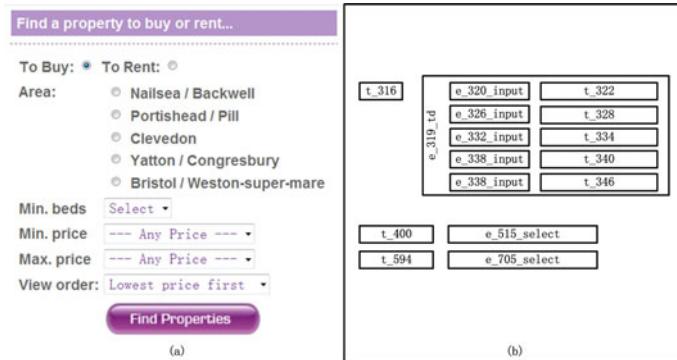


Fig. 2. Form on Heritage with its Page Model

HTML elements. The page model is also expanded with linguistic annotations and machine learning based classifications, relying on domain-specific knowledge. Secondly, the segmentation model describes conceptual relationships between form elements and associates them with texts that are potential labels. This is obtained from the page model through the segmentation mapping, which employs a number of heuristics to build a conceptual segmentation by choosing proper structural and visual relationships among fields and texts. Thirdly, the domain model describes the form as a tree of domain-specific elements, such as price elements. We construct it by joining the domain annotations from the page model with the segmentation results from the segmentation model. The joining process, called phenomenological mapping, is guided by the domain ontology. To adapt OPAL for another domain, one only needs to configure the relevant annotations, define domain specific elements, and instantiate rules for the phenomenological mappings from a set of common templates.

In the following paragraphs, we discuss the OPAL system using <http://www.heritage4homes.co.uk> as our running example (see Figure 2).

3.1 Page Model

The page model represents the structural, textual, and visual properties of the web page as rendered by a browser engine. We represent the DOM tree (including element, text, comment, and attribute) in structural relations and encode the tree structure using the start/end encoding, see Figure 2.

Relying on domain-specific knowledge, the page model is expanded with linguistic annotations and machine learning based classifications on texts appearing on web pages.

3.2 Segmentation Model

Taking the page model, the segmentation mapping labels and groups form elements, such as fields and texts. This mapping exploits heuristics on structural and visual properties and yields the segmentation model. Groups are derived

from similarities in the HTML design and in the visual appearance of form fields. A consecutive list of segments makes a group (parent segment) if they satisfy the similarity conditions and their least common ancestor contains no other segments. We translate this and other similar conditions into rules as shown below (where E_S refers to a list of segments).

```

group( $E_S$ )  $\Leftarrow$  similarFieldSequence( $E_S$ )  $\wedge$  leastCommonAnc( $A, E_S$ )  $\wedge$ 
2      not hasAdditionalField( $A, E_S$ ).
leastCommonAnc( $A, E_S$ )  $\Leftarrow$  commonAnc( $A, E_S$ )  $\wedge$  not(child( $C, A$ )  $\wedge$  commonAnc( $C, E_S$ )).
4 partOf( $E, A$ )  $\Leftarrow$  group( $E_S$ )  $\wedge$  member( $E, E_S$ )  $\wedge$  leastCommonAnc( $A, E_S$ ).

```

Labeling, e.g. label assignment to form segments, is achieved through three heuristics, such that `hasLabel(E, L, T)` is true if segment E is assigned with label node L which contains text T . The three heuristics are as follows: (1) *HTML label*, which extracts HTML labels from web forms. (2) *Greatest unique ancestor*, which finds the greatest unique ancestor of a segment and associates all its text descendants with the segment. (3) *Segment alignment*, which does label assignment based on the position of segment and text members. In the last heuristics, we identify the list of all text groups in the parent segment, partitioned by each occurring child segment. If an interleaving situation is encountered, we perform one-to-one assignments between text groups to child segments. We show a fragment of the results produced by the three heuristics below.

```

hasLabel( $e\_320\_input, t\_322, "Nailsea / Backwell"$ ).
2 hasLabel( $e\_326\_input, t\_328, "Portishead / Pill"$ ).
hasLabel( $e\_515\_select, t\_400, "Min Price"$ ).
4 hasLabel( $e\_705\_select, t\_594, "Max Price"$ ).
hasLabel( $e\_319\_td, t\_316, "Area"$ ).

```

3.3 Domain Model

This model describes conceptual entities on forms as defined in the domain ontology. The ontology provides a reference description of how such forms occur in practice on the considered websites. Figure 3 presents three relevant fragments of our ontology for the UK real estate domain. (A) the top concepts defining how a real-estate web form is constructed, (B) the price segments, modeling the structure of a price input/selection facility, and (C) the area/branch segment describing a search facility based on location information, i.e. a geographic area (e.g. London). In the ontology, classes (or concepts) are represented as unary first-order predicates, denoting the type of an object in the form. The “part of” relation encodes the hierarchical structure of the form segments. Attributes are represented as binary relations between concepts and DOM nodes. For example, it may represent the fact that a price field is a minimum/maximum price field or that an order-by input field is ordering in ascending or descending order. Furthermore, additional constraints are modeled for attributes (defining domains for attribute values) and relations (cardinality and compatibility constraints). To discuss the ontology, we explain (B) as an example. A price segment is composed by an optional currency element and one or more price elements. The price-Type

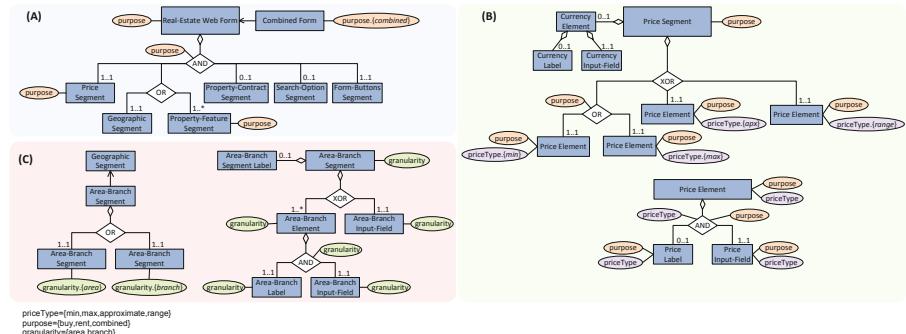


Fig. 3. The Real-Estate Web-Form Ontology (fragment)

attribute is used to denote different price-input facilities occurring in real-estate web forms, e.g. price range selection, a pair composed by a minimum and a maximum price, etc. In case of such a range, the price elements must agree on the value of the purpose attribute (the compatibility constraint). Each price element consists of a label and a price input field.

From the ontology, we derive a phenomenological mapping to classify the form elements and derive the domain form model. This mapping is called *phenomenological* as it connects the abstract concepts of the ontology (e.g. price element) with observable phenomena on the web pages, e.g., a select box with a text label. For example, we annotated “Nailsea” as a location in the extended page model and obtain the result that “Nailsea / Backwell” is associated with the first radio button in the “Area” segment. Hence we conclude that this radio button should be classified as an area/branch element. In this fashion, we associate an area/branch element to the five radio buttons. The second and third dropdown lists are identified as price elements with minimum and maximum for price-type value. It is interesting to note that in some cases, the annotations for form elements can be associated with existing DOM nodes, while in other cases—when a corresponding HTML element does not exist—the phenomenological rules generate an artificial bounding box for such elements which is then annotated using the ontology.

Experiments. We conducted experiments for OPAL up to the segmentation model on a publicly available dataset (ICQ dataset) to test its domain independence and the complete OPAL on a sampled UK real-estate dataset to show the enhancement of ontology. In the former case, we covered 100 query interfaces from 5 domains and achieved 94% F1-score for field labeling and 93% for correct segmentation. For the latter case, where there are 50 randomly selected UK real-estate web sites, we achieved over 97% in field labeling and 95% for the segmentation.

4 Ontologies for Object Recognition and Analysis

We introduce AMBER (“*Adaptable Model-based Extraction of Result Pages*”) to identify result page records and extract them as objects. AMBER is parameterized with a domain ontology which models knowledge on **(i)** records and attributes of the domain, **(ii)** low-level (textual) representations of these concepts, and **(iii)** constraints linking representations to records and attributes. Parametrized with these constraints, domain-independent heuristics exploit the repeated structure of a result page to derive attributes and records. AMBER is implemented to allow an explicit formulation of the heuristics and easy adaptation to different domains.

4.1 Background

There have been a number of approaches for extracting records from result pages, but they were mostly either semi-automated or domain-independent, as surveyed e.g. in [8,9]. In contrast, and as in case of web form analysis, we follow a domain-aware approach: Based on domain-specific annotations in the result page, e.g. marking all occurring rent prices, we identify the occurring data areas, segment the records, and align the attributes of the found records.

Our approach works in four steps: During the **(i) retrieval** and **(ii) annotation stage**, the *page* and *annotation model* are obtained to represent the DOM of a live browser and relevant domain-specific markup. **(iii) The phenomenological mapping** constructs an *attribute model* which summarizes the annotations into potential record attributes occurring on the analyzed web page. **(iv) The segmentation mapping** uses the structural and visual information from the browser model and the attributes identified in the attribute model to locate data areas and segment these areas into individual data records. As a result, we obtain the *result page model* for the given page.

4.2 Algorithm Description

Due to different representations for the same content on different web sites, automatic data extraction usually results in a complex and time-consuming task. Existing approaches mostly try to detect those repeated structural patterns in the DOM tree that represent data records. This approach has the advantage of being domain independent because it relies only on the structural similarities between different records (within the same or among different pages). However, we can safely say that all past *domain-independent* attempts describing *all* possibly occurring page structures have failed.

In our approach, we combine the detection of repeated structures with background knowledge of the domain. We provide the analysis process with a semantic description of the data that we expect to find on the web page, plus a set of “constraints” that are known to hold in the domain. Our experiments show that this combination results in a much more precise analysis and enables a simple consistency check over the extracted data.

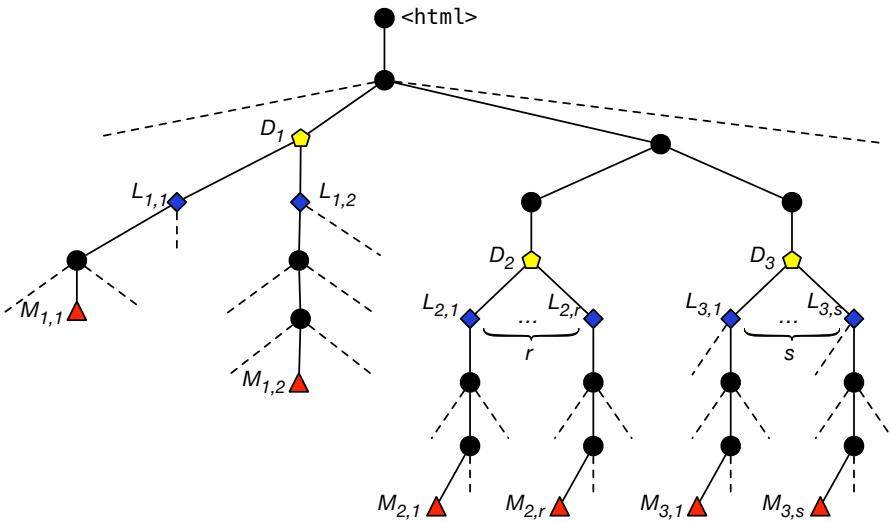


Fig. 4. Results on Zoopla

Data Area Identification. A data area in a result page is identified by leveraging *mandatory elements*. A mandatory element is a domain concept that appears in all records of a given data area, e.g., the location in a real-estate website. Since in this phase the records are yet to be discovered, the mandatory elements are identified by matching the content of the text nodes with a domain ontology.

Since the matching process is intrinsically imperfect, we allow false-positives during the identification of mandatory nodes. To reduce the false-positives among the matched mandatory nodes (MD-nodes), we group MD-nodes with same (or similar) *depth* in the DOM and similar *relative position* among their siblings. We then consider only MD-nodes belonging to the largest group and discard other nodes. The least common ancestor in the DOM tree of all identified MD-nodes is considered the data area root.

Because it is possible that a result page contains several data areas, we repeat the data area identification process and choose the largest group as a data area until the largest group contains only one record (probably noise) or we eliminate all groups. In Figure 4, D_1 and D_2 are data areas.

Record Segmentation. The records within a data area are identified as sub-trees rooted at children of the data area root. The segmentation process uses *record separators*, i.e., sub-trees interleaved with records, typically containing neither text nor a URL. As a first step, each subtree in the DOM containing a single MD-node and rooted at a direct child of the data area root is considered a *candidate record*. A subsequent step tries to “expand” the candidate record to adjacent subtrees in the DOM. We therefore consider the siblings of the candidate record. If they are record separators, we consider each candidate record as a proper record; otherwise we apply the following steps: (1) Compute the *distance l* between two candidate records as the number of siblings between their root

nodes. **(2)** Consider all possible $2 \times (l - 1)$ offsets for record boundaries, compute the similarity between the identified records, and choose the one with highest similarity. **(3)** Whenever several expansions have the same similarity, we choose the one with the highest structural similarity among records. To break ties, we pick the one delimited by the largest number of record separators.

Data Alignment. After the segmentation phase, it is possible to have inconsistent mappings between the intended record structure and the DOM tree. For example, we might have multiple assignments for functional attributes (e.g., two prices for the same apartment) or we might miss the assignment for a mandatory attribute (e.g., apartments with no rooms). This is due to the unavoidable uncertainty introduced by the textual annotations and by the heuristic analysis rules. In these cases some form of data-reconciliation process must be applied. Since data cleaning and reconciliation is out of the scope of this work, we rely on state-of-the-art techniques for these tasks [10]. In particular, since we already exploit domain knowledge during the analysis, we will leverage on the constraints of the result-page model and of the domain ontology to disambiguate multiple assignments and identify missing attributes.

When it is not possible to use background knowledge to disambiguate a multiple assignment, we adopt a scoring mechanism that takes into account the position within the record of the node associated to the attribute's value and the length (in characters) of the value. In particular we privilege nodes at the beginning of the records (considering the DOM document order) and nodes with synthetic content (i.e., the text-node length). The reason is that meaningful content usually appears in the top left corner of the records (i.e., the corresponding nodes will appear early in document order) and they appear in the form of synthetic text.

4.3 Evaluation

We report on our preliminary statistical analysis of the current state of result pages in this section. We illustrate the result of an *experimental evaluation* of the AMBER approach. AMBER has been evaluated on 50 randomly selected UK real-estate web sites from 2,810 UK real-estate web sites from yellow page. In order to evaluate both precision and recall of our technique, after randomly chosen 50 web sites, we chose one or two result pages from each site (some sites have only one result page), and annotated them manually. For each result page, we annotated both the position and content of data area, the position and content of each record and the position and content of each data attributes.

AMBER reaches 100% on both precision and recall of 126 data areas, and 100% precision and 99.8% recall for 2101 records, 99.4% precision and 99.0% recall for 6733 data attributes. While this result itself shows the effectiveness of AMBER, it is worth noting that our program could not perfectly identify all records on only 2 web sites. Although the extraction of data attributes is based on a raw, incomplete ontology, we still achieve a very high rate of 99.4% precision and 99.0% recall for 6733 data attributes.

5 Web Scale Extraction with OXPath

OXPath is the DIADEM formalism for automating web extraction tasks, capable of scaling to the size of the web. A properly specified OXPath expression outputs *structured web objects* adhering to knowledge encoded in domain-specific ontologies consistent with a known phenomenology.

Extracting and aggregating web information is not a new challenge. Previous approaches, in the overwhelming majority, either (1) require service providers to deliver their data in a structured fashion (e.g. the Semantic Web); or, (2) “wrap” unstructured information sources to extract and aggregate relevant data. The first case levies requirements that service providers have little incentive to adopt, which leaves us with wrapping. Wrapping a website, however, is often tedious, since many AJAX-enabled web applications reveal the relevant data only through user interactions. Previous work does not adequately address web page scripting. Even when scripting is considered, the simulation of user actions is neither declarative nor succinct, but rather relies on imperative scripts.

5.1 Language

OXPath extends XPath 1.0 with four conceptual extensions: Actions to navigate the user interface of web applications, exposure to rendered visual information, extraction markers to specify data to extract, and the Kleene star to facilitate iteration over a set of pages with an unknown extent.

Actions. For simulating user actions such as clicks or mouse-overs, OXPath introduces *contextual*, as in `{click}`, and *absolute action steps* with a trailing slash, as in `{click /}`. Since actions may modify or replace the DOM, we assume that they always return a new DOM. Absolute actions return DOM roots, contextual actions return the nodes in the new DOM matched by the *action-free prefix* of the performed action, which is obtained from the segment starting at the previous absolute action by dropping all intermediate contextual actions and extraction markers.

Style Axis and Visible Field Access. We introduce two extensions for lightweight visual navigation: a new axis for accessing CSS DOM node properties and a new node test for selecting only visible form fields. The `style` axis navigates the actual CSS properties as returned by the DOM `style` object. For example, it is possible to select nodes based on their (rendered) color or font size.

To ease field navigation, OXPath introduces the node-test `field()`, which relies on the `style` axis to access the computed CSS style to exclude fields that are not visible, e.g., `/descendant::field()[1]` selects the first visible field in document order.

Extraction Marker. In OXPath, we introduce a new kind of qualifier, the *extraction marker*, to identify nodes as representatives for records as well as to form attributes for these records. For example,

```

1 doc("news.google.com")//div[contains(@class,"story")]:<story>
2     [.//h2:<title=string(.)>]
     [.//span[style::color="#767676"]:<source=string(.)>]

```

extracts a `story` element for each current story on Google News, along with its title and sources (as strings), producing:

```

<story><title>Tax cuts ...</title>
2   <source>Washington Post</source>
   <source>Wall Street Journal</source> ... </story>

```

The nesting in the result mirrors the structure of the OXPath expression: extraction markers in a predicate (`title`, `source`) represent attributes to the last marker outside the predicate (`story`).

Kleene Star Finally, we add the Kleene star, as in [11]. For example, the following expression queries Google for “Oxford”, traverses all accessible result pages and extracts all links.

```

doc("google.com")/descendant::field()[1]/{"Oxford"}
2   /following::field()[1]/{click /}
   /(&descendant::a:<Link=(@href)>[.#= "Next"]/{click /})*

```

To limit the range of the Kleene star, one can specify upper and lower bounds on the multiplicity, e.g., `(...)*{3,8}`.

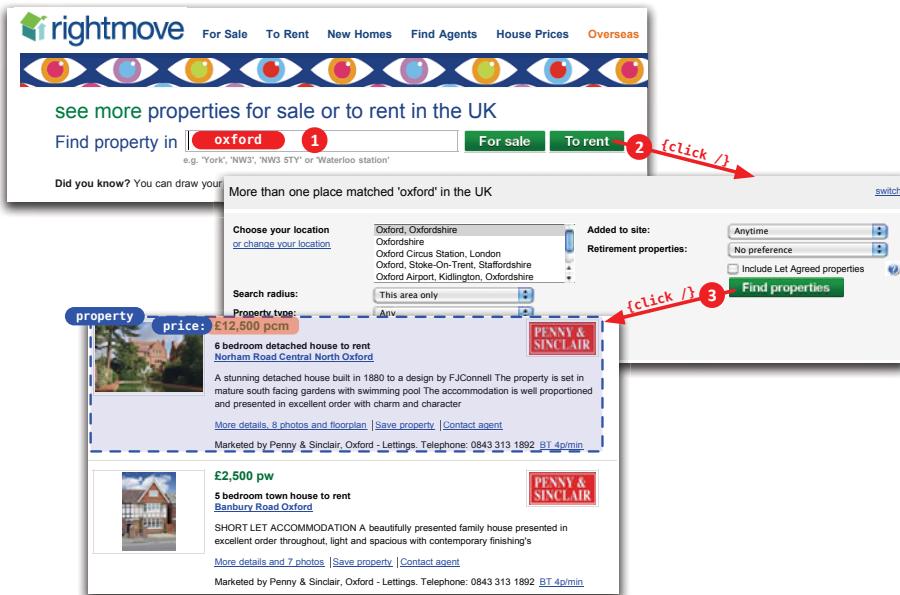
5.2 Example Expression

The majority of OXPath notation is familiar to XPath users. In the previous section, we carefully extend XPath to achieve desired automation and extraction features. Consider now a full expression, shown in Figure 5. In this example, we define an OXPath expression that extracts prices of properties from `rightmove.co.uk`. We begin in line 1 by retrieving the first HTML page via the `doc(url)` function, which OXPath borrows from XQuery. We continue through line 3 by *simulating* user action for many different form input fields, spread over multiple HTML pages. Note here that OXPath allows the use of the CSS selectors `#` and `..`, which allows selection of nodes based on their `id` and `class` attributes, respectively. Line 4 uses the Kleene star to specify extraction from all possible result pages, which are traversed by clicking on hyperlinks containing the word “next”. Finally, line 5 identifies all relevant properties and extracts their corresponding prices. This example could be extended to encapsulate all attributes relevant to each found web object, which in this example are all rentable properties from this site that satisfy our criteria.

5.3 System

OXPath uses a three layer architecture as shown in Figure 6:

- (1) The **Web Access Layer** enables programmatic access to a page’s DOM as rendered by a modern web browser. This is required to evaluate OXPath expressions which interact with web applications or access computed CSS style



```

doc("rightmove.co.uk")/descendant::field()[1]/{"Oxford"}
2 //following::input#rent/{click()}/select#minBedrooms>{"2"/}
//select#maxPrice>{"1,750 PCM"}/input#submit/{click()}/
4 //(/a[contains(., "next")]/{click()})*
//ol#summaries/li:<property>[//p.price:<price=string(.)>];

```

Fig. 5. OXPath for Rental Properties in Oxford

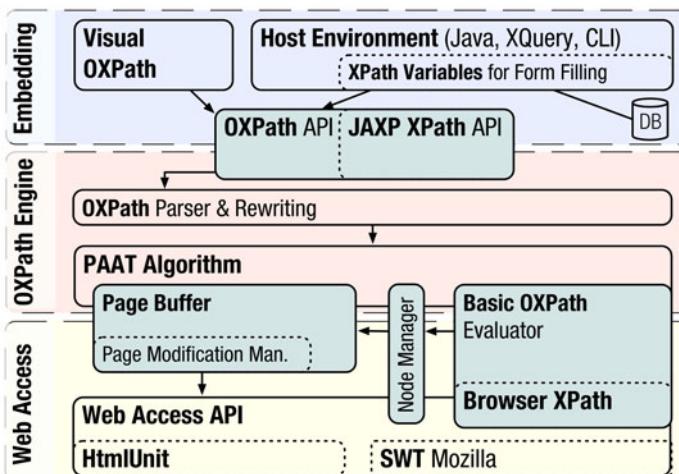


Fig. 6. OXPath System Architecture

information. The web layer is implemented as a facade which promotes interchangeability of the underlying browser engine (Firefox and HTMLUnit).

(2) The **Engine Layer** evaluates OXPath expressions. Basic OXPath steps, i.e., subexpressions without actions, extraction markers and Kleene stars, are directly handled by the browser's XPath engine.

(3) The **Embedding Layer** facilitates the integration of OXPath within other systems and provides a host environment to instantiate OXPath expressions. The host environment provides variable bindings from databases, files, or even other OXPath expressions for use within OXPath. To facilitate OXPath integration, we slightly extend the JAXP API to provide an output stream for extracted data.

Though OXPath can be used in any number of host languages such as Java, XSLT, or Ruby, we designed a lightweight host language for large-scale data extraction. It is a Pig Latin[12]-like query language with OXPath subqueries, grouping, and aggregation. We separate these tasks, as well as the provision of variable bindings, from the core language to preserve the declarative nature of OXPath and to guarantee efficient evaluation of the core features.

OXPath is complemented by a visual user interface, a Firefox add-on that records mouse clicks, keystrokes, and other actions in sequence to construct an equivalent OXPath expression. It also allows the selection and annotation of nodes used to construct a generalised extraction expression. We are actively improving the visual interface and developing a visual debugger for OXPath.

5.4 Further Reading

We have limited the scope of the discussion here to fundamental aspects of the OXPath formalism. For further details, please see [13]. In particular, this work introduces the PAAT (page-at-a-time) algorithm that evaluates OXPath expressions with intelligent page caching without sacrificing the efficiency of regular XPath. In this way, PAAT guarantees polynomial evaluation time and memory use independent of the number of visited pages. Further, this work highlights experimental results of the current prototype. These experimental results validate our strong theoretical time and memory guarantees. OXPath performs faster than comparable systems by at least an order of magnitude in experiments where a constant memory footprint for OXPath can be empirically observed. No observed competitor managed memory as intelligently as PAAT: either all target pages were cached (requiring linear memory w.r.t. pages visited) or a fixed number of pages were cached (requiring pages to be revisited in the general case). For example applications of OXPath over real-world web data, please see [14,15].

References

1. Nguyen, H., Nguyen, T., Freire, J.: Learning to Extract From Labels. In: Proc. of the VLDB Endowment (VLDB), pp. 684–694 (2008)
2. Su, W., Wang, J., Lochovsky, F.H.: ODE: Ontology-Assisted Data Extraction. ACM Transactions on Database Systems 34(2) (2009)

3. Kushmerick, N.: Learning to invoke web forms. In: Chung, S., Schmidt, D.C. (eds.) CoopIS 2003, DOA 2003, and ODBASE 2003. LNCS, vol. 2888, pp. 997–1013. Springer, Heidelberg (2003)
4. Shadbolt, N., Hall, W., Berners-Lee, T.: The Semantic Web Revisited. IEEE Intelligent Systems 21(3), 96–101 (2006)
5. Wu, W., Doan, A., Yu, C., Meng, W.: Modeling and Extracting Deep-Web Query Interfaces. In: Advances in Information & Intelligent Systems, pp. 65–90 (2009)
6. Dragut, E.C., Kabisch, T., Yu, C., Leser, U.: A Hierarchical Approach to Model Web Query Interfaces for Web Source Integration. In: Proc. Int'l. Conf. on Very Large Data Bases (VLDB), pp. 325–336 (2009)
7. Raghavan, S., Garcia-Molina, H.: Crawling the Hidden Web. In: Proc. Int'l. Conf. on Very Large Data Bases (VLDB), pp. 129–138 (2001)
8. Chang, C.-H., Kayed, M., Grgis, M.R., Shaalan, K.F.: A survey of web information extraction systems. IEEE Trans. Knowl. Data Eng. 18(10), 1411–1428 (2006)
9. Laender, A.H.F., Ribeiro-Neto, B.A., da Silva, A.S., Teixeira, J.S.: A brief survey of web data extraction tools. SIGMOD Record 31(2), 84–93 (2002)
10. Batini, C., Scannapieco, M.: Data Quality: Concepts, Methodologies and Techniques. Springer, Heidelberg (2006)
11. Marx, M.: Conditional xpath. ACM Trans. Database Syst. 30(4), 929–959 (2005)
12. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, pp. 1099–1110. ACM, New York (2008)
13. Furche, T., Gottlob, G., Grasso, G., Schallhart, C., Sellers, A.J.: Oxpath: A language for scalable, memory-efficient data extraction from web applications. In: Proc. of the VLDB Endowment PVLDB (2011) (to appear)
14. Sellers, A., Furche, T., Gottlob, G., Grasso, G., Schallhart, C.: Taking the oxpath down the deep web. In: Proceedings of the 14th International Conference on Extending Database Technology, EDBT/ICDT 2011, pp. 542–545. ACM, New York (2011)
15. Sellers, A.J., Furche, T., Gottlob, G., Grasso, G., Schallhart, C.: Oxpath: little language, little memory, great value. In: Proceedings of the 20th International Conference Companion on World Wide Web, WWW 2011, pp. 261–264. ACM, New York (2011)

Analyzing Cross-System User Modeling on the Social Web

Fabian Abel, Samur Araújo, Qi Gao, and Geert-Jan Houben

Web Information Systems, Delft University of Technology

{f.abel,s.f.cardosodearaujo,q.gao,g.j.p.m.houben}@tudelft.nl

Abstract. In this article, we analyze tag-based user profiles, which result from social tagging activities in Social Web systems and particularly in Flickr, Twitter and Delicious. We investigate the characteristics of tag-based user profiles within these systems, examine to what extent tag-based profiles of individual users overlap between the systems and identify significant benefits of cross-system user modeling by means of aggregating the different profiles of a same user.

We present a set of cross-system user modeling strategies and evaluate their performance in generating valuable profiles in the context of tag and resource recommendations in Flickr, Twitter and Delicious. Our evaluation shows that the cross-system user modeling strategies outperform other strategies significantly and have tremendous impact on the recommendation quality in cold-start settings where systems have sparse information about their users.

1 Introduction

Social tagging comes in different flavors. Social bookmarking systems like Delicious allow people to tag their bookmarks, in photo sharing platforms such as Flickr users annotate images, and in micro-blogging systems like Twitter people can assign so-called hash tags to their posts. In the last decade we saw a variety of research efforts in the field of social tagging systems ranging from the analysis of social tagging structures [9,7], via information retrieval in folksonomy systems [4,11], to personalization [16,17]. However, most studies have been conducted in the context of particular systems, for example, analyzing the tagging behavior within Delicious [7] or computing tag recommendations in Flickr based on the tagging characteristics within the Flickr system [17]. Our research complements these studies and investigates tagging behavior and user modeling across system boundaries to support engineering of Social Web systems that aim for personalization.

We therefore identified users, who have an account at Flickr, Twitter *and* Delicious, and crawled more than 2 million tagging activities which were performed by these users in the three systems. Based on this dataset, we study the tagging behavior of the same user in different systems. We furthermore present and analyze various cross-system user modeling strategies. For example, we show that it is possible to generate tag-based profiles based on Twitter activities for improving tag and bookmark recommendations in Delicious.

1.1 Personomies and Tag-Based Profiles

The emerging structure that evolves from social tagging is called folksonomy. A folksonomy is basically a set of tag assignments, user-tag-resource bindings attached with some timestamp that indicates when a tag assignment was performed. We base our research on the folksonomy model $\mathbb{F} = (U, T, R, Y)$ proposed by Hotho et. al [11], where U , T , R and Y refer to the sets of users, tags, resources and tag assignments respectively. We also model Twitter posts by means of tag assignments, i.e. we consider each post (*tweet*) as a resource and hash tags such as “#icwe2011”, which appear in the post, are—without the “#” symbol—treated as tags. Given the folksonomy model \mathbb{F} , the user-specific part of a folksonomy, the *personomy*, can be defined as follows (cf. [11]).

Definition 1 (Personomy). *The personomy $\mathbb{P}_u = (T_u, R_u, Y_u)$ of a given user $u \in U$ is the restriction of \mathbb{F} to u , where T_u and R_u are finite sets of tags and resources respectively that are referenced from tag assignments Y_u performed by the user u .*

While the personomy specifies the tag assignments that were actually performed by a specific user, the *tag-based profile* $P(u)$ is an abstraction of the user that represents the user as a set of weighted tags (cf. [8,15]).

Definition 2 (Tag-based profile). *The tag-based profile of a user u is a set of weighted tags where the weight of a tag t is computed by a certain strategy w with respect to the given user u .*

$$P(u) = \{(t, w(u, t)) | t \in T_{source}, u \in U\} \quad (1)$$

$w(u, t)$ is the weight that is associated with tag t for a given user u . T_{source} is the source set of tags from which tags are extracted for the tag-based profile $P(u)$.

The weights associated with the tags in a tag-based profile $P(u)$ do not necessarily correspond to the tag assignments in the user’s personomy \mathbb{P}_u . For example, $P(u)$ may also specify the weight for a tag t_i that does neither occur in the personomy \mathbb{P}_u nor in the folksonomy \mathbb{F} , i.e. where $t_i \notin T_u \wedge t_i \notin T$. With $P(u)@k$ we describe the subset of a tag-based profile $P(u)$ that contains the k tag-weight pairs that have the highest weights. $\bar{P}(u)$ denotes the tag-based profile for which the weights are normalized so that the sum of all weights in $P(u)$ is equal to 1, and with $|P(u)|$ we refer to the number of distinct tags contained in the tag-based profile.

1.2 Problem Definition

Today, individual users may be active in various social tagging systems so that tag-based profiles about the same user are distributed across different systems. How do tag-based profiles of the same user differ from system to system? To what extent do these profiles overlap? Can cross-system user modeling support the construction of tag-based profiles and how does it impact the performance of recommender systems in cold-start situations where user profiles are sparse? In this paper we answer these questions and tackle the following research challenge.

Definition 3 (User modeling challenge). *Given a user u , the user modeling strategies have to construct a tag-based profile $P(u)$ so that the performance of tag and resource recommendations in cold-start situations is maximized.*

Following related work (e.g. [17]) we define the recommendation tasks as ranking problems:

Definition 4 (Cold-start Recommendation challenge). *Given a tag-based user profile $P(u)$, the personomy of the user $\mathbb{P}_{u,target} = (T_u, R_u, Y_u)$ and a set of tags T_{target} and a set of resources R_{target} , which are not explicitly connected to u ($T_u \cap T_{target} = \emptyset$ and $R_u \cap R_{target} = \emptyset$), the challenge of the recommendation strategies is to rank those tags $t \in T_{target}$ and resources $r \in R_{target}$ so that tags/resources that are most relevant to the user u appear at the very top of the ranking.*

For both tag and resource recommendations, we do not aim to optimize the recommender algorithm itself, but we identify those user modeling strategies that support the recommender algorithms best.

1.3 Related Work

Today, several research efforts aim to support re-use of user profile data on the Social Web. With standardization of exchange languages (e.g. FOAF¹) and APIs (e.g. OpenSocial²), standardized authentication protocols such as OpenID³, solutions for identifying users across systems [6] and research on generic user modeling [13,10,3], cross-system user modeling becomes tangible. In previous work we analyzed the nature of social networking profiles on the Social Web and introduced a service for aggregating tag-based profiles [2]. Szomszor et al. presented an approach to merge user's tag clouds from two different social networking websites to generate richer user interest profiles [18], but did not investigate the impact of the generated profiles on personalization.

Different from personalization in social tagging systems that targets single systems [16,17], cross-system personalization makes the investments in personalizing a system transferable to other systems. Mehta et al. showed that cross-system personalization makes recommender systems more robust against spam and cold start problems [14]. However, Mehta et al. could not test their approaches on Social Web data where individual user interactions are performed across different systems and domains, but experiments have been conducted on user data, which originated from one system and was split to simulate different systems. In contrast to that, in this paper we evaluate cross-system user modeling and its impact on cold-start recommendations on real world datasets from three typical Social Web systems: Flickr, Twitter and Delicious.

¹ <http://xmlns.com/foaf/spec/>

² <http://code.google.com/apis/opensocial/>

³ <http://openid.net/>

2 User Modeling Strategies

The cross-system user modeling strategies that we discuss in this paper consist of the following building blocks: (1) source of user data, (2) semantic enrichment, (3) weighting scheme.

2.1 Source of User Data

In order to construct a user profile and adapt functionality to the individual users, systems require information about their users [12]. The data that is exploited to create user profiles (cf. T_{source} in Definition 2) might come from different sources. In our evaluation (see Section 4) we will therefore compare the following sources of information.

Target Personomy Tags $T_{\mathbb{P}_{target}}$. The traditional approach to user modeling is to exploit the user-specific activities observed in the target system that aims for personalization. In a folksonomy system one would thus exploit the personomy of the user \mathbb{P}_u (see Definition 1) that is inferred from the folksonomy \mathbb{F}_{target} of the target system.

Target Folksonomy Tags $T_{\mathbb{F}_{target}}$. If the personomy of the user \mathbb{P}_u is rather sparse or even empty, one has to find other sources of information that are applicable to generate a user profile. Therefore, we define another baseline strategy that considers all activities performed in the target system as if they would have been performed by the given user. Hence, by considering the complete folksonomy \mathbb{F}_{target} one obtains some sort of *average profile* which promotes these tags that are popular for the complete folksonomy.

Foreign Personomy Tags $T_{\mathbb{P}_{foreign}}$. If the user's personomy in the target system is sparse or empty then another strategy is to utilize the personomy from another system. For example, if a user u has not annotated any resource in Delicious yet, this strategy harnesses the hash tags the user assigned to her Twitter posts, i.e. u 's personomy from Twitter.

2.2 Semantic Enrichment

Semantics of tag-based profiles are not well defined. For example, tag-based recommender systems may encounter situations where relevant resources are discarded because tags assigned to these resources are syntactically different from the user profile tags: $P(u)$ may contain the tag “bloggingstuff” while relevant resources are tagged with “blog”, “weblog” or “blogging”. To counter such problems we analyze different strategies that further enrich tag-based user profiles and the set of tags T_{source} particularly.

Tag Similarity χ_{sim} . This strategy enriches the initial set of tags T_{source} with tags that have high string similarity to one of the tags in T_{source} . In our analysis we apply Jaro-Winkler distance [19] for computing the similarity between two tags. It considers the number of matching characters, number of transformations that would be required to unify the two strings, as well

as the length of both tags and ranges between 0 (no similarity) and 1 (exact match). For example, the Jaro-Winkler distance between “blogging” and “blogging-stuff” is 0.95.

Cross-system Rules χ_{rules} . Cross-system user modeling has to deal with heterogeneous vocabularies (cf. Section 3): the overlap of tags between different folksonomies may be low and tags that are popular in one system might be unpopular in another system [2,5]. Cross-system rules enrich tag-based profiles based on association rules deduced from characteristics observed across two systems. These association rules can be phrased as follows.

If tag t_a occurs in the personomy $\mathbb{P}_{u,A}$ of user u in folksonomy system A then tag t_b occurs in u ’s personomy $\mathbb{P}_{u,B}$ in system B .

In our cross-system user modeling analysis we exploit association rules that are applicable to the tags T_{source} of a given profile, i.e. $t_a \in T_{source}$. We add tags t_b to T_{source} that are generated by those association rules that have the highest confidence while satisfying a minimum support of 0.1.

We use $\chi_{sim}@k$ and $\chi_{rules}@k$ to refer to these strategies that add the top k most appropriate (based on similarity and confidence respectively) tags to T_{source} .

2.3 Weighting Scheme

We compare different approaches to determine the weight $w(u, t)$ associated with a tag $t \in T_{source}$ in the tag-based profile of a specific user u (see Definition 2).

Term frequency TF . For the given set of tags T_{source} and a given set of tag assignments Y_{source} , the term frequency corresponds to the relative number of tag assignments in Y_{source} that refer to the tag $t \in T_{source}$.

$$w_{TF}(u, t) = \frac{|\{r \in R | (u, t, r) \in Y_{source}\}|}{|Y_{source}|} \quad (2)$$

TF and Inverse Document Frequency $TF \times IDF$. The inverse document frequency (IDF) can be applied to value the term specificity. From the perspective of the user modeling strategies, IDF refers to the inverse fraction of the number of distinct users that applied a given tag $t \in T_{source}$.

$$w_{TF \times IDF}(u, t) = w_{TF}(u, t) \cdot \log\left(\frac{|U|}{|\{u_i \in U | (u_i, t, r) \in Y_{source}\}|}\right) \quad (3)$$

The weighting schemes thus require a set of tag assignments Y_{source} as input. In accordance to the source of user data, we obtain Y_{source} either from the target personomy \mathbb{P}_{target} , from the target folksonomy \mathbb{F}_{target} or from the foreign personomy $\mathbb{P}_{foreign}$. For example, $TF_{\mathbb{F}_{target}}$ considers all tag assignments from the target folksonomy \mathbb{F}_{target} .

2.4 Assembling User Modeling Strategies

The actual user modeling strategy for constructing the tag-based profile is built by combining (i) a data source with (ii) some semantic enrichment method, and

Table 1. Example user modeling strategies based on TF weighting scheme, semantic enrichment and different data sources (target folksonomy \mathbb{F}_{target} or foreign persononomy $\mathbb{P}_{foreign}$)

Strategy	Source	Enrich	Weighting	Cross?	Description
$P_{T@F_t, TF@F_t}$ (baseline)	$T_{\mathbb{F}_{target}}$	–	$TF_{\mathbb{F}_{target}}$	–	Tags and term frequency weights are taken from the target folksonomy. The tag-based profile thus contains the most popular tags.
$P_{T@F_t, TF@P_f}$	$T_{\mathbb{F}_{target}}$	–	$TF_{\mathbb{P}_{foreign}}$	+	While the actual tags in the profile are obtained from the target folksonomy, the weights are computed based on the user-specific persononomy from a foreign system.
$P_{T@P_f, TF@F_t, \chi_s}$	$T_{\mathbb{P}_{foreign}}$	χ_{sim}	$TF_{\mathbb{F}_{target}}$	+	Tag-based profile contains user-specific tags from foreign persononomy enriched with tag similarity. Weights are taken from the global term frequency in \mathbb{F}_{target} .
$P_{T@P_f, TF@P_f, \chi_r}$	$T_{\mathbb{P}_{foreign}}$	χ_{rules}	$TF_{\mathbb{P}_{foreign}}$	+	Tag-based profile is constructed based on user-specific persononomy as available in the foreign system enriched with cross-system rule.

(iii) a weighting scheme. The semantic enrichment is an optional feature. For example, $P_{T@P_t, TF@P_t}$ corresponds to the tag-based profile proposed in [8,15], which exploits the user-specific persononomy in the target system. $P_{T@P_t, TF@P_t}$ would not be applicable in cold-start situations where new users register to a system as the target persononomy \mathbb{P}_{target} would be empty in such situations. Table 1 lists some strategies that can be applied in cold-start situations. For example, $P_{T@P_f, TF@F_t, \chi_s}$ denotes the strategy that utilizes the persononomy $\mathbb{P}_{foreign}$ from a foreign system as data source, enriches the profile using tag similarity, and computes weights according to the global term frequency in the target folksonomy.

Based on the different combinations of data sources and weighting schemes and the additional option of semantic enrichment we obtain a powerful framework for generating cross-system user modeling strategies.

3 Analysis of Tag-Based Profiles on the Social Web

Before evaluating the user modeling strategies presented in the previous section, it is important to study the nature of persononomies and of the corresponding tag-based profiles ($P_{T@P_t, TF@P_t}$) distributed across the different Social Web systems. In particular, we approach the following research questions.

1. What are the characteristics of the individual tag-based profiles in Twitter, Flickr and Delicious?
2. How do the tag-based profiles of individual users overlap between the different systems?

3.1 Data Collection

To investigate the questions above, we crawled public profiles of 421188 distinct users via the Social Graph API⁴, which makes information about connections between user accounts available via its Web service. By exploiting Google profiles

⁴ <http://code.google.com/apis/socialgraph/>

Table 2. Tagging statistics of the (a) Twitter-Delicious dataset (1500 users) and (b) Flickr-Delicious dataset (1467 users)

(a) Twitter-Delicious (1500 users)				(b) Flickr-Delicious (1467 users)			
	Twitter	Delicious	Aggregated		Flickr	Delicious	Aggregated
distinct tags	25668	72901	91515	distinct tags	72671	59275	119056
TAS	80464	619856	700320	TAS	892378	683665	1576043
distinct tags/user	26.1	180.06	191.88	distinct tags/user	109.44	189.92	292.63
TAS/user	53.64	413.24	1466.88	TAS/user	608.30	466.03	1074.33
tagged resources	57236	124520	181756	tagged resources	166423	109242	272701
resources per user	38.99	91.13	130.12	resources per user	113.45	85.07	198.50

of users, who interlinked their different online accounts, the API provides the list of accounts associated with a particular user.

For our experiments we were interested in users having accounts at Twitter, Delicious and Flickr. Given the 421188 users, 2007 users linked their Twitter and Delicious accounts and 1500 of these users applied tags in both systems. Table 2(a) lists the corresponding tagging statistics of these users. Accordingly, Table 2(b) shows the number of tags, resources, and tag assignments (TAS) of the 1467 users, who linked their Flickr and Delicious profiles. We make both datasets available online [1].

3.2 Tag-Based Profiles within Systems

By nature, Twitter is not a typical social tagging system. However, it enables users to annotate posts by means of *hash tags*, which start with the “#” symbol. Table 2(a) indicates that people make use of this tagging feature. On average, each user performed 53.64 (hash) tag assignments on 38.99 Twitter posts using 26.1 distinct (hash) tags. In Delicious, the same users are tagging more frequently with an average of 413.24 tag assignments and an average number of 180.06 distinct tags per user profile. Further, users assign, on average, 4.53 tags to each of their Delicious bookmark while they only attach 1.38 hash tags to their Twitter posts.

The second dataset obtained from those users, who linked their Flickr and Delicious profiles, shows similar statistics for the Delicious profiles (see Table 2(b)). Further, it is interesting to see that users perform, on average, 142.27 tag assignments more in Flickr than they do in Delicious. However, at the same time their Flickr profile contains less distinct tags (109.44) than their Delicious profiles (189.92). The variety of tags in the Delicious profiles is thus higher than the ones in Flickr profiles, which might make them more valuable for personalization (see Section 4).

Figure 1 describes the above observation in detail: less than 20% of the tag-based Flickr profiles contain more than 200 tags and less than 5% of the Twitter profiles contain more than 100 distinct (hash) tags. By contrast, more than 40% of the Delicious profiles have more than 200 tags.

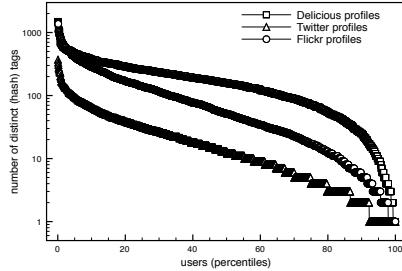


Fig. 1. Distinct tags per tag-based Twitter, Flickr and Delicious profile

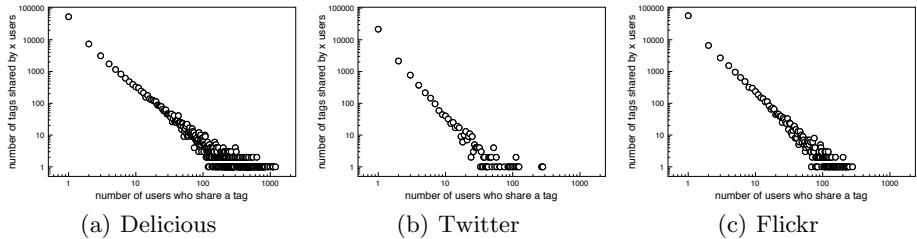


Fig. 2. Homogeneity of tag-based profiles: number of tags that occur in a certain number of (a) Delicious, (b) Twitter, and (c) Flickr profiles

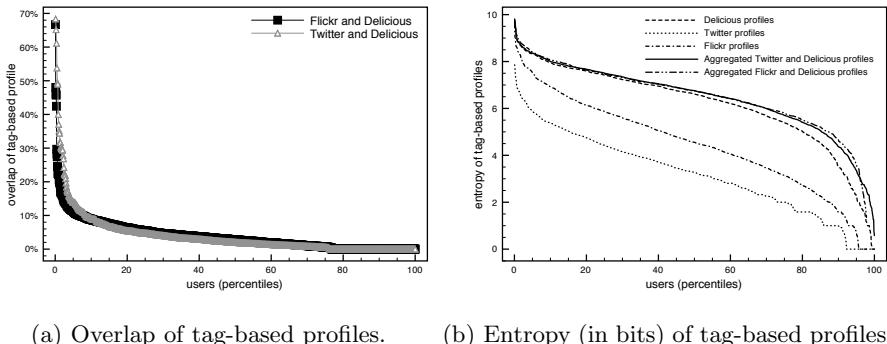


Fig. 3. Characteristics of tag-based profiles: (a) overlap of tag-based Twitter and Delicious profiles as well as Flickr and Delicious profiles and (b) entropy (in bits) of tag-based profiles: cross-system user modeling by means of profile aggregation increases entropy of tag-based profiles

Figure 2 describes the tag usage and therefore the homogeneity of the profiles in the three different systems. In all three systems, most tags (e.g., more than 50000 tags in Delicious) only occur in exactly one profile and only a few single tags occur in more than 100 profiles. For example, there exists only one tag, namely “design”, that occurs in more than 80% of the Delicious profiles.

3.3 Tag-Based Profiles across Systems

In the context of cross-system user modeling the overlap of tag-based profiles is of particular interest. If the tagging behavior of an individual user differs from system to system then the overlap of the user's tag-based profiles will be small and the corresponding tag-based profiles will probably reveal complementary facets for characterizing the user. Figure 3(a) shows to which degree the tag-based profiles of the individual users overlap between the different systems. We see that the overlap between tag-based Twitter and Delicious profiles of individual users as well as the overlap between the Flickr and Delicious profiles is very small. In fact, for less than 10% of the users, the Twitter or Flickr profiles have an overlap of more than 10% with the tag-based Delicious profiles. This observation indicates that the re-use of tag-based profiles for solving the user modeling challenge posed at the beginning of this paper is not trivial at all and has to be done in an intelligent way. For example, simply re-using a tag-based Twitter profile in Delicious would reflect only a very small part of the user's characteristics in Delicious.

The small overlap of the tag-based profiles implicates that the profiles of an individual user at different Social Web platforms reveal different characteristics of the user. Cross-system user modeling by means of profile aggregation thus allows for more valuable profiles with regards to the information about the user. In Figure 3(b) we compare the entropy of tag-based profiles. Entropy quantifies the information embodied in a tag-based profile $P(u)$, which specifies a weight for each tag $t \in T_{source}$ (see Definition 2), and is computed as follows.

$$\text{entropy}(P(u)) = \sum_{t \in T_{source}} p(t) \cdot (-\log_2(p(t))) \quad (4)$$

In Equation 4, $p(t)$ denotes the probability that the tag t was utilized by the corresponding user and can be modeled via the weight $w(u, t)$ from the normalized tag-based profile $\bar{P}(u)$, for which the sum of all weights is equal to 1.

Figure 3(b) shows that tag-based profiles in Delicious have higher entropy than the ones in Twitter and Flickr – even though Flickr features the highest number of tag assignments per user (cf. Table 2(b)). Thus, the variety of tags in Delicious profiles is higher than in Flickr profiles. We further aggregated the profiles of the individual users by accumulating their tag-based profiles from Twitter and Delicious as well as Flickr and Delicious. For both types of profiles entropy increases significantly with respect to the service-specific profiles. Hence, cross-system user modeling based on profile aggregation has significant impact on the entropy of tag-based profiles.

3.4 Synopsis

From our analysis of tag-based profiles ($P_{T@P_t, TF@P_t}$) we conclude that users reveal different profile facets in the different systems on the Social Web. Tag-based profiles of same users in different systems overlap only little. For more than 90% of the users, the tag-based profiles obtained from Delicious, Twitter and Flickr overlap to less than 10%. Cross-system user modeling allows for more

valuable profiles. By aggregating tag-based profiles, information gain and entropy improve significantly.

4 Analysis of Cross-System User Modeling Strategies

The above results reveal already benefits of cross-system user modeling for tag-based profiles. We now evaluate the performance of the cross-system user modeling strategies for supporting tag and bookmark recommendations in cold-start situations (see Definition 3) and investigate the following research questions.

1. Which user modeling strategies generate the most valuable tag-based profiles for recommending tags and resources to users in which context?
2. How do the different building blocks of the user modeling strategies (e.g. source of user data) influence the quality of the tag-based profiles?

4.1 Experimental Setup

The actual recommendation algorithm incorporates a user profile $P(u)$, as delivered by the given user modeling strategy, to compute a ranking of items (tags and resources) so that items relevant to the user u appear at the top of the ranking. Therefore, we specify a generic algorithm that can be customized with specific ranking and user modeling strategies.

Generic Recommendation Algorithm *The generic recommendation algorithm recommend(u, s, um) computes a ranked list of entities appropriate to a user u by exploiting a given ranking strategy s and a given user modeling strategy um .*

1. **Input:** ranking strategy s , user modeling strategy um , user u
2. $P(u) = um.modelUser(u)$ (compute user profile)
3. $\tau = s.rank(P(u))$ (rank entities w.r.t. $P(u)$)
4. **Output:** τ (ranked list of entities)

As we do not aim to optimize the ranking strategy, but want to investigate the impact of the different cross-system user modeling strategies on the recommendation performance, we deliberately apply lightweight algorithms for ranking the tags and resources respectively.

Tag Recommendation. Regarding the tag recommendation task, our goal is to point the user to tags she has not applied yet, but might be of interest for her and are worthwhile to explore. Hence, we filter the tag-based profile and remove those tags that already occur in the personomy of the user u in the target system ($\mathbb{P}_{u,target}$). The remaining tags are then ranked according to the weight that is associated with the tags in the tag-based profile $P(u)$. If two tags have the same weight then the order of these tags is chosen randomly.

Resource Recommendation. We apply cosine similarity to generate resource recommendations. Given a tag-based user profile $P(u)$, the corresponding

personomy of the user $\mathbb{P}_{u,target} = (T_u, R_u, Y_u)$, the folksonomy of the target system $\mathbb{F}_{target} = (T, R, Y)$, and a set of resources $R_{target} \subseteq R$, which are not explicitly connected to u ($R_u \cap R_{target} = \emptyset$), we first generate tag-based profiles for each resource $r \in R_{target}$. The tag-based resource profile resembles the tag-based user profile specified in Definition 2: $P(r) = \{\{t, w(r, t)\} | t \in T, r \in R_{target}\}$. In accordance to the *TF* weighting scheme applied by the user modeling strategies, we compute the weights associated with the resources as follows.

$$w_{TF}(r, t) = \frac{|\{u \in U | (u, t, r) \in Y\}|}{|\{u \in U, t_x \in T | (u, t_x, r) \in Y\}|} \quad (5)$$

Given the target folksonomy \mathbb{F}_{target} , the weight $w(r, t)$ is thus given by the number of users who assigned tag t to resource r divided by the overall number of tag assignments attached to r . To compute cosine similarity, the tag-based profiles of the user ($P(u)$) and resource ($P(r)$) are represented in a vector space model via \mathbf{u} and \mathbf{r} respectively where each dimension of these vectors corresponds to a certain tag t that occurs in both profiles. Finally, each resource $r \in R_{target}$ is ranked according its cosine similarity with the user profile representation \mathbf{u} so that those resources that have high similarity occur at the top of the ranking.

Evaluation Methodology. We evaluate the quality of the recommender algorithms and therewith the user modeling strategies respectively by means of a *leave-many-out evaluation*. For simulating a cold-start where a new or yet unknown user u is interested in recommendations in system A , we first remove all tag assignments Y_u performed by u in system A from the folksonomy. Each recommender strategy then has to compute a ranked list of recommendations. The quality of the recommendations is measured via *MRR* (Mean Reciprocal Rank), which indicates at which rank the first *relevant* item occurs on average, *S@k* (success at rank k), which stands for the mean probability that a *relevant* item occurs within the top k of the ranking, and *P@k* (precision at rank k), which represents the average proportion of *relevant* items within the top k (cf. [17]). We consider only these items as relevant to which the user u actually referred to in the tag assignments Y_u that were removed before computing the recommendations. For a given user modeling strategy, we ran the experiment for each individual user of our datasets (cf. Section 3). We tested the statistical significance of our results with a two-tailed *t*-Test where the significance level was set to $\alpha = 0.01$.

4.2 Results: Cold-Start Tag Recommendations

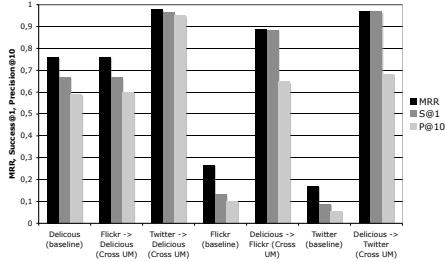
Table 3 gives a detailed overview on the performance of the different user modeling strategies for recommending *tags* in Twitter and Delicious. The best strategies (emphasized) benefit from cross-system user modeling and outperform the other strategies significantly. For example, the precision of the top ten tag recommendations in Delicious based on tags obtained from the user's personomy in Twitter ($T_{\mathbb{P}_{foreign}}$), which are weighted according to the global tag frequencies obtained from the Delicious folksonomy ($TF_{\mathbb{F}_{target}}$), is 94.4% ($P_{T@P_f, TF@F_t}$ strategy) and therewith improves by more than 90% over the $P_{T@F_t, TF@F_t}$ baseline

Table 3. User modeling performances measured via tag recommendation quality in a cold-start setting in Twitter and Delicious: tags that occur in the tag-based profile are either selected from the folksonomy of the target system (T_{target}) or from the user-specific personony in the foreign system ($T_{\mathbb{P}_{\text{foreign}}}$) whereas the weighting scheme is term frequency (TF) or TF multiplied by inverse document frequency ($TF \times IDF$), which is either computed based on the target folksonomy (TF_{target}) or foreign personony ($TF_{\mathbb{P}_{\text{foreign}}}$). “Cross?” denotes whether the corresponding strategy benefits from cross-system user modeling (+) or not (-). Semantic enrichment (Enrich) is either based on tag similarity (χ_{sim}), cross-system association rules (χ_{rules}) or not applied (-).

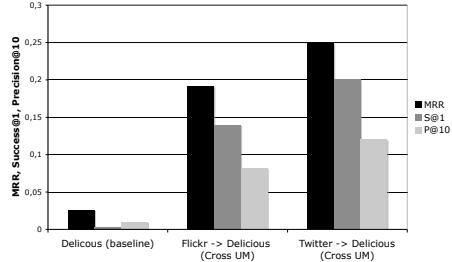
Strategy	Source	Enrich	Weighting	Cross?	MRR	S@1	S@10	P@5	P@10	P@20
Twitter (foreign) → Delicious (target):										
$P_{\text{T@P}_f, \text{TF@F}_t}$	$T_{\mathbb{P}_{\text{foreign}}}$	-	$TF_{\mathbb{P}_{\text{target}}}$	+	0.979	96.6	99.6	95.1	94.4	91.4
$P_{\text{T@P}_f, \text{TF@P}_f}$	$T_{\mathbb{P}_{\text{foreign}}}$	-	$TF_{\mathbb{P}_{\text{foreign}}}$	+	0.631	43.7	97.7	52.4	62.1	72.5
$P_{\text{T@F}_t, \text{TF@P}_f}$	$T_{\mathbb{P}_{\text{target}}}$	-	$TF_{\mathbb{P}_{\text{target}}}$	-	0.764	69.1	94.5	50.8	49.5	48.4
$P_{\text{T@F}_t, \text{TF@P}_f}$	$T_{\mathbb{P}_{\text{target}}}$	-	$TF_{\mathbb{P}_{\text{foreign}}}$	+	0.623	51.5	85.3	34.1	28.5	26.4
$P_{\text{T@P}_f, \text{TF} \times \text{IDF@F}_t}$	$T_{\mathbb{P}_{\text{foreign}}}$	-	$TF \times IDF_{\text{target}}$	+	0.974	95.6	99.6	95.3	94.4	91.5
$P_{\text{T@P}_f, \text{TF} \times \text{IDF@P}_f}$	$T_{\mathbb{P}_{\text{foreign}}}$	-	$TF \times IDF_{\mathbb{P}_{\text{foreign}}}$	+	0.650	4.5	97.8	52.8	62.1	72.4
$P_{\text{T@F}_t, \text{TF} \times \text{IDF@F}_t}$	$T_{\mathbb{P}_{\text{target}}}$	-	$TF \times IDF_{\text{target}}$	-	0.244	4.5	88.0	12.5	28.9	33.9
$P_{\text{T@F}_t, \text{TF} \times \text{IDF@P}_f}$	$T_{\mathbb{P}_{\text{target}}}$	-	$TF \times IDF_{\mathbb{P}_{\text{foreign}}}$	+	0.608	49.2	85.1	33.4	28.3	26.4
$P_{\text{T@P}_f, \text{TF}@F_t, \chi_s}$	$T_{\mathbb{P}_{\text{foreign}}}$	χ_{sim}	$TF_{\mathbb{P}_{\text{target}}}$	+	0.978	96.4	99.6	94.1	93.8	91.0
$P_{\text{T@P}_f, \text{TF}@P_f, \chi_s}$	$T_{\mathbb{P}_{\text{foreign}}}$	χ_{sim}	$TF_{\mathbb{P}_{\text{foreign}}}$	+	0.631	43.7	97.7	52.4	62.1	72.6
$P_{\text{T@P}_f, \text{TF}@F_t, \chi_r}$	$T_{\mathbb{P}_{\text{foreign}}}$	χ_{rules}	$TF_{\mathbb{P}_{\text{target}}}$	+	0.778	69.1	93.3	71.0	68.5	63.1
$P_{\text{T@P}_f, \text{TF}@P_f, \chi_r}$	$T_{\mathbb{P}_{\text{foreign}}}$	χ_{rules}	$TF_{\mathbb{P}_{\text{foreign}}}$	+	0.624	43.7	96.9	53.2	61.4	71.5
Delicious (foreign) → Twitter (target):										
$P_{\text{T@P}_f, \text{TF}@F_t}$	$T_{\mathbb{P}_{\text{foreign}}}$	-	$TF_{\mathbb{P}_{\text{target}}}$	+	0.194	9.7	39.9	10.1	10.7	11.2
$P_{\text{T@P}_f, \text{TF}@P_f}$	$T_{\mathbb{P}_{\text{foreign}}}$	-	$TF_{\mathbb{P}_{\text{foreign}}}$	+	0.970	96.9	96.9	82.1	64.0	42.2
$P_{\text{T@F}_t, \text{TF}@P_f}$	$T_{\mathbb{P}_{\text{target}}}$	-	$TF_{\mathbb{P}_{\text{target}}}$	-	0.102	5.9	20.4	3.4	3.2	2.9
$P_{\text{T@F}_t, \text{TF}@P_f}$	$T_{\mathbb{P}_{\text{target}}}$	-	$TF_{\mathbb{P}_{\text{foreign}}}$	+	0.749	74.8	74.8	45.5	29.5	17.0
$P_{\text{T@P}_f, \text{TF} \times \text{IDF}@P_f}$	$T_{\mathbb{P}_{\text{foreign}}}$	-	$TF \times IDF_{\mathbb{P}_{\text{target}}}$	+	0.683	60.5	84.4	33.7	27.2	21.6
$P_{\text{T@P}_f, \text{TF} \times \text{IDF}@P_f}$	$T_{\mathbb{P}_{\text{foreign}}}$	-	$TF \times IDF_{\mathbb{P}_{\text{foreign}}}$	+	0.386	27.5	60.9	19.9	16.8	14.5
$P_{\text{T@F}_t, \text{TF} \times \text{IDF}@P_f}$	$T_{\mathbb{P}_{\text{target}}}$	-	$TF \times IDF_{\mathbb{P}_{\text{target}}}$	-	0.171	8.5	33.7	4.8	5.4	5.8
$P_{\text{T@F}_t, \text{TF} \times \text{IDF}@P_f}$	$T_{\mathbb{P}_{\text{target}}}$	-	$TF \times IDF_{\mathbb{P}_{\text{foreign}}}$	+	0.209	11.8	38.0	8.5	6.7	5.0
$P_{\text{T@P}_f, \text{TF}@F_t, \chi_s}$	$T_{\mathbb{P}_{\text{foreign}}}$	χ_{sim}	$TF_{\mathbb{P}_{\text{target}}}$	+	0.194	9.7	39.7	10.1	10.7	11.2
$P_{\text{T@P}_f, \text{TF}@P_f, \chi_s}$	$T_{\mathbb{P}_{\text{foreign}}}$	χ_{sim}	$TF_{\mathbb{P}_{\text{foreign}}}$	+	0.970	96.9	96.9	82.1	64.0	42.1
$P_{\text{T@P}_f, \text{TF}@F_t, \chi_r}$	$T_{\mathbb{P}_{\text{foreign}}}$	χ_{rules}	$TF_{\mathbb{P}_{\text{target}}}$	+	0.096	5.9	18.8	3.9	5.2	4.2
$P_{\text{T@P}_f, \text{TF}@P_f, \chi_r}$	$T_{\mathbb{P}_{\text{foreign}}}$	χ_{rules}	$TF_{\mathbb{P}_{\text{foreign}}}$	+	0.969	96.9	96.9	83.8	68.3	47.2

strategy, which does not make use of cross-system user modeling and achieves a precision of 49.5.

This improvement is even significantly higher for recommending tags in Twitter where the best cross-system user modeling strategy ($P_{\text{T@P}_f, \text{TF}@P_f, \chi_r}$) achieves 68.3 for $P@10$ in comparison to 5.4, achieved by the best non-cross-system strategy. Overall, tag recommendations in Twitter seem to be more difficult: while the best strategies allow for high success values (e.g., $S@1 > 95$), it requires advanced user modeling methods like the application of cross-system association rules (χ_{rules}) to obtain high precisions for $P@10$ and $P@20$. Possible explanations for this can be derived from Section 3: Twitter user profiles are more sparse than the corresponding Delicious profiles and exhibit a lower entropy. This might also explain why $TF_{\mathbb{P}_{\text{foreign}}}$, i.e. weighting based on tag-based profile in Delicious, is more successful in Twitter than $TF_{\mathbb{P}_{\text{target}}}$, the Twitter-based weighting scheme. Hence, when applying the user modeling strategies in a cross-system setting, it is important to consider the general characteristics of the tagging data available in the different systems (as we did in Section 3). For example, the consideration



(a) Tag recommendations.



(b) Resource recommendations.

Fig. 4. Comparison of different cross-system user modeling settings in the context of (a) tag recommendations and (b) resource recommendations. The baseline strategies follow the $P_{T@F_t, TF@F_t}$ approach, $P_{T@F_f, TF@P_f}$ is applied for settings when the target is Delicious and $P_{T@P_f, TF@P_f, \chi_r}$ is used for recommendations in Flickr and Twitter.

of Delicious profile data leads to higher improvements than the consideration of Twitter profile data. However, independent from the setting we see that cross-system user modeling has significant impact on the recommendation quality.

Our evaluations on the Flickr-Delicious dataset confirmed the results listed in Table 3 with two remarkable differences that are summarized in Figure 4(a): (1) Flickr-based cross-system user modeling (*Flickr → Delicious (Cross UM)*) for tag recommendations in Delicious improves the quality only slightly (no significant difference) and (2) in the context of tag recommendations in Flickr, cross-system user modeling leads again to tremendous improvements, however, the quality gain is not as high as for the *Delicious → Twitter* setting. The absolute tag recommendation performances are however high for all cross-system settings and work best for the interplay of Delicious and Twitter (e.g. *MRR > 0.95*) as depicted in Figure 4(a). Overall, we conclude that the significant improvements of the tag recommendation performance further support the utility of cross-system user modeling.

4.3 Results: Cold-Start Resource Recommendations

Table 4 overviews the performance of the different user modeling strategies for cold-start bookmark recommendations in Delicious. This task is more difficult than the tag recommendation task as the items, which are recommended to the user, are not directly connected to the user, but indirectly via the tags [16]. Further, the fraction of relevant items is much smaller: given an average number of 91.3 relevant bookmarks per user, the probability to randomly select a resource, which will be bookmarked by the user, from the set of 124520 available resources is 0.0007 ($P@1$). With this in mind, it is interesting to see that the cross-system user modeling strategies, which fully exploit the Twitter persononomies (source selection and weighting is based on $\mathbb{P}_{foreign}$), succeed for recommending Delicious bookmarks in a cold-start setting and perform significantly better than all other strategies (see Table 4).

Table 4. User modeling performances measured via Delicious bookmark recommendation quality in a cold-start setting with Twitter: tags that occur in the tag-based profile are either selected from the folksonomy of Delicious ($T_{F_{target}}$) or from the user-specific personomy in Twitter ($T_{P_{foreign}}$) whereas the weighting scheme is the term frequency (TF) or TF multiplied by inverse document frequency ($TF \times IDF$), which is either computed based on the target Delicious folksonomy ($TF_{F_{target}}$) or foreign personomy available in Twitter ($TF_{P_{foreign}}$). “Cross?” denotes whether the corresponding strategy benefits from cross-system user modeling (+) or not (-). Semantic enrichment (Enrich) is either based on tag similarity (χ_{sim}), cross-system association rules (χ_{rules}) or not applied (-).

Strategy	Source	Enrich	Weighting	Cross?	MRR	S@1	S@10	P@5	P@10	P@20
$P_{T@P_f, TF@F_t}$	$T_{P_{foreign}}$	-	$TF_{F_{target}}$	+	0.026	0.3	5.6	0.7	0.6	0.5
$P_{T@P_f, TF@P_f}$	$T_{P_{foreign}}$	-	$TF_{P_{foreign}}$	+	0.191	14.4	26.0	11.7	9.3	7.7
$F_{T@F_t, TF@F_t}$	$T_{F_{target}}$	-	$TF_{F_{target}}$	-	0.026	0.3	7.8	1.0	0.9	0.6
$P_{T@F_t, TF@P_f}$	$T_{F_{target}}$	-	$TF_{P_{foreign}}$	+	0.055	2.3	11.1	2.1	1.9	2.1
$P_{T@P_f, TFxIDF@F_t}$	$T_{P_{foreign}}$	-	$TF \times IDF_{F_{target}}$	+	2.7	0.5	3.9	0.6	0.4	0.6
$P_{T@P_f, TFxIDF@P_f}$	$T_{P_{foreign}}$	-	$TF \times IDF_{P_{foreign}}$	+	0.244	18.9	32.2	14.5	11.6	9.4
$F_{T@F_t, TFxIDF@F_t}$	$T_{F_{target}}$	-	$TF \times IDF_{F_{target}}$	-	0.006	0.1	1.5	0.1	0.1	0.1
$P_{T@F_t, TFxIDF@P_f}$	$T_{F_{target}}$	-	$TF \times IDF_{P_{foreign}}$	+	0.053	2.3	10.9	2.0	2.3	2.2
$P_{T@P_f, TFxIDF@F_t, \chi_s}$	$T_{P_{foreign}}$	χ_{sim}	$TF \times IDF_{F_{target}}$	+	0.026	0.5	3.9	0.5	0.4	0.6
$P_{T@P_f, TFxIDF@P_f, \chi_s}$	$T_{P_{foreign}}$	χ_{sim}	$TF \times IDF_{P_{foreign}}$	+	0.244	18.7	32.4	14.5	11.7	9.4
$P_{T@P_f, TFxIDF@F_t, \chi_r}$	$T_{P_{foreign}}$	χ_{rules}	$TF \times IDF_{F_{target}}$	+	0.01	0.1	1.3	0.1	0.2	0.2
$P_{T@P_f, TFxIDF@P_f, \chi_r}$	$T_{P_{foreign}}$	χ_{rules}	$TF \times IDF_{P_{foreign}}$	+	0.252	20.1	33.2	14.9	12.1	10.0

While the TF weighting scheme is best applicable in the tag recommendation context, computing weights based on $TF \times IDF$ and $TF \times IDF_{P_{foreign}}$ in particular perform best for the Delicious bookmark recommendations. We further discover that the semantic enrichment by means of cross-system association rules, that indicate which tags are likely to occur in the tag-based Delicious profile if a user applied certain tags in Twitter, has significant impact on the quality of the generated profiles. The difference in performance of $P_{T@P_f, TFxIDF@P_f, \chi_r}$, which is again the best strategy, with respect to the other strategies is statistically significant.

These findings can also be confirmed on the setting of Flickr and Delicious. Figure 4(b) compares the performance of the cross-system user modeling strategy $P_{T@P_f, TFxIDF@P_f, \chi_r}$ with the best baseline strategy that does not consider external profile information. Exploiting Twitter for resource recommendations in Delicious is more appropriate than the consideration of Flickr profiles. For example, given the Twitter-based profile, the probability to recommend a relevant resource ($S@1$) is 0.2 in comparison with 0.14 for the Flickr-based profiles. Further, with Twitter-based cross-system user modeling we achieve a precision ($P@10$) of 0.12, which is more than 10 times higher than the baseline strategy.

4.4 Synopsis

For both tag and resource recommendations, we conclude that cross-system user modeling strategies generate significantly more valuable tag-based profiles than those strategies which do not consider external profile information. We see that the interplay between Twitter and Delicious is more successful than the one of

Flickr and Delicious. The selection of the building blocks for composing an appropriate cross-system user modeling strategy impacts the quality significantly. For example, in the resource recommendation context those user modeling strategies perform best that fully exploit personomies from other systems (source selection and weighting is based on $\mathbb{P}_{foreign}$).

5 Conclusions

In this paper we analyzed strategies for modeling users across Social Web system boundaries to infer insights for engineers of Web applications that aim for personalization in cold-start situations. We investigated the characteristics of tag-based profiles that result from tagging activities in Flickr, Twitter and Delicious, and discovered that the tag-based profiles which a same user has at different platforms only overlap little. Cross-system user modeling by means of aggregating the user-specific profiles from the different platforms results in significantly more valuable profiles with respect to entropy. Within the scope of tag and resource recommendations in cold-start settings we further revealed that cross-system user modeling strategies have significant impact on the performance of the recommendation quality. For example, user modeling based on external personomies results in a more than 10 times higher precision for recommending Delicious bookmarks.

Acknowledgements. This work is partially sponsored by the EU FP7 project ImREAL (<http://imreal-project.eu>).

References

1. Abel, F., Araújo, S., Gao, Q., Houben, G.J., Tao, K.: Supporting website: datasets and further details (2011), <http://wis.ewi.tudelft.nl/icwe2011/um/>
2. Abel, F., Henze, N., Herder, E., Krause, D.: Interweaving public user profiles on the web. In: De Bra, P., Kobsa, A., Chin, D. (eds.) UMAP 2010. LNCS, vol. 6075, pp. 16–27. Springer, Heidelberg (2010)
3. Aroyo, L., Dolog, P., Houben, G.J., Kravcik, M., Naeve, A., Nilsson, M., Wild, F.: Interoperability in pesonalized adaptive learning. J. Educational Technology & Society 9 (2), 4–18 (2006)
4. Bao, S., Xue, G., Wu, X., Yu, Y., Fei, B., Su, Z.: Optimizing Web Search using Social Annotations. In: Proc. of 16th Int. World Wide Web Conference (WWW 2007), pp. 501–510. ACM Press, New York (2007)
5. Bischoff, K., Fir醤, C., Paiu, R., Nejdl, W.: Can All Tags Be Used for Search? In: Proc. of Conf. on Information and Knowledge Management (CIKM 2008). ACM, New York (2008)
6. Carmagnola, F., Cena, F.: User identification for cross-system personalisation. Information Sciences: an International Journal 179(1-2), 16–32 (2009)
7. Dellschaft, K., Staab, S.: An epistemic dynamic model for tagging systems. In: Brusilovsky, P., Davis, H.C. (eds.) Proceedings of the 19th ACM Conference on Hypertext and Hypermedia (HT 2008), pp. 71–80. ACM, Pittsburgh (2008)
8. Fir醤, C.S., Nejdl, W., Paiu, R.: The Benefit of Using Tag-based Profiles. In: Proc. of 2007 Latin American Web Conference (LA-WEB 2007), pp. 32–41. IEEE Computer Society, Washington, DC, USA (2007)

9. Golder, S.A., Huberman, B.A.: Usage patterns of collaborative tagging systems. *Journal of Information Science* 32(2), 198–208 (2006)
10. Heckmann, D., Schwartz, T., Brandherm, B., Schmitz, M., von Wilamowitz-Moellendorff, M.: GUMO – The General User Model Ontology. In: Ardissono, L., Brna, P., Mitrović, A. (eds.) UM 2005. LNCS (LNAI), vol. 3538, pp. 428–432. Springer, Heidelberg (2005)
11. Hotho, A., Jäschke, R., Schmitz, C., Stumme, G.: Information retrieval in folksonomies: Search and ranking. In: Proc. of the 3rd European Semantic Web Conference (ESWC 2010), Budva, Montenegro, pp. 411–426. Springer, Heidelberg (2006)
12. Jameson, A.: Adaptive interfaces and agents. In: The HCI Handbook: Fundamentals, Evolving Technologies and Emerging Applications, pp. 305–330 (2003)
13. Kobsa, A.: Generic user modeling systems. *User Modeling and User-Adapted Interaction* 11(1-2), 49–63 (2001)
14. Mehta, B.: Cross System Personalization: Enabling personalization across multiple systems. VDM Verlag, Saarbrücken (2009)
15. Michlmayr, E., Cayzer, S.: Learning User Profiles from Tagging Data and Leveraging them for Personal(ized) Information Access. In: Proc. of the WWW 2007 Workshop on Tagging and Metadata for Social Information Organization (2007)
16. Sen, S., Vig, J., Riedl, J.: Tagommenders: connecting users to items through tags. In: Quemada, J., León, G., Maarek, Y.S., Nejdl, W. (eds.) Proceedings of the 18th International Conference on World Wide Web (WWW 2009), pp. 671–680. ACM, New York (2009)
17. Sigurbjörnsson, B., van Zwol, R.: Flickr tag recommendation based on collective knowledge. In: Proc. of 17th Int. World Wide Web Conference (WWW 2008), pp. 327–336. ACM Press, New York (2008)
18. Szomszor, M., Alani, H., Cantador, I., O’Hara, K., Shadbolt, N.: Semantic modelling of user interests based on cross-folksonomy analysis. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T.W., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 632–648. Springer, Heidelberg (2008)
19. Winkler, W.E.: The State of Record Linkage and Current Research Problems. Technical report, Statistical Research Division, U.S. Census Bureau (1999)

Parallel Data Access for Multiway Rank Joins

Adnan Abid and Marco Tagliasacchi

Dipartimento di Elettronica e Informazione – Politecnico di Milano,
Piazza Leonardo da Vinci, 32 – 20133 Milano, Italy
{abid,tagliasa}@elet.polimi.it

Abstract. Rank join operators perform a relational join among two or more relations, assign numeric scores to the join results based on the given scoring function and return K join results with the highest scores. The top- K join results are obtained by accessing a subset of data from the input relations. This paper addresses the problem of getting top- K join results from two or more *search* services which can be accessed in parallel, and are characterized by non negligible response times. The objectives are: *i*) minimize the time to get top- K join results. *ii*) avoid the access to the data that does not contribute to the top- K join results.

This paper proposes a multi-way rank join operator that achieves the above mentioned objectives by using a score guided data pulling strategy. This strategy minimizes the time to get top- K join results by extracting data in parallel from all Web services, while it also avoids accessing the data that is not useful to compute top- K join results, by pausing and resuming the data access from different Web services adaptively, based on the observed score values of the retrieved tuples. An extensive experimental study evaluates the performance of the proposed approach and shows that it minimizes the time to get top- K join results, while incurring few extra data accesses, as compared to the state of the art rank join operators.

Keywords: rank joins, rank queries, score guided data pulling, top- K queries.

1 Introduction

Rank join operators have a widespread applicability in many application domains. Hence, a set of specialized rank join operators have been recently proposed in the literature [1][4][6][8][9]. These operators are capable of producing top- K join results by accessing a subset of data from each source, provided the score aggregation function is monotone, and the data retrieved from each source is sorted in descending order of score.

As an illustrative example, consider a person who wants to plan his visit to Paris by searching for a good quality hotel and a restaurant, which are situated close to each other and are highly recommended by their customers. This can be accomplished by extracting information from suitable data sources available on the Web and merging the information to get the top rated resultant combinations, as contemplated in Search Computing [3]. The Web services, e.g. Yahoo!

`Local` or `yelp.com`, can be used to find the places of interest in a city. The data can be processed to produce the top- K scoring join results of hotels and restaurants. A sample rank query based on the above example is the following:

```
SELECT h.name, r.name, 0.6*h.rating+0.4*r.rating as score
FROM Hotels h, Restaurants r
WHERE h.zip = r.zip AND h.city= ‘Paris’ AND r.city = ‘Paris’
RANK BY 0.6*h.rating+0.4*r.rating
```

Motivation: The recent solutions to rank join problem [5][7][12] focus on providing instance optimal algorithms regarding the I/O cost. The I/O cost is a quantity proportional to the overall number of fetched tuples. So these algorithms minimize the total number of tuples to be accessed in order to find the top- K join results. *Hash Rank Join (HRJN*)* [7] is an instance optimal algorithm in terms of I/O cost and it introduces a physical rank join operator. This algorithm has been further improved in [5] and [12]. Indeed, this optimization of the I/O cost helps reducing the total time to compute the top- K join results as well, yet total time can be further reduced for the following reason: these I/O optimal algorithms access data from the data sources in a serial manner, i.e. they access data from one source, process it and then fetch the data from the next *most suitable* source. The latter is selected based on a *pulling strategy*, which determines the source to be accessed to, in order to minimize the I/O cost. However, in the context of using Web services as data sources, data processing time is found to be negligible as compared to data fetching time. So, most of the time is spent in waiting for retrieving the data. Therefore, an alternative approach that extracts data from all data sources in parallel should be used in order to reduce the data extraction time from all sources by overlapping the waiting times. This calls for a parallel data access strategy.

A simple parallel strategy keeps on extracting data from each Web service in parallel until top- K join results can be reported. We call this strategy *Parallel Rank Join (PRJ)*. As an illustrative example, assume that we can extract top 10 join results from 2 different Web services after fetching 3 data pages from each Web service. Figure 1 shows the behaviour of both HRJN* and PRJ. It can be observed that both HRJN* and PRJ approaches have shortcomings: HRJN* takes a large amount of time to complete, whereas, PRJ costs more in terms of I/O as it may retrieve unnecessary data (e.g. C4 and C5). This requires the design of a rank join operator that is specifically conceived to meet the objectives of getting top- K join results quickly and restricting access to unwanted data, when using Web services or similar data sources.

As a contribution we propose a *Controlled Parallel Rank Join (cPRJ)* algorithm that computes the top- K join results from multiple Web services with a *controlled* parallel data access which minimizes both total time, and the I/O cost, to report top- K join results. In Section 2 we provide the preliminaries. The algorithm is explained in Section 3. A variant of the algorithm is presented in Section 4. The experiments and results are discussed in Section 5, and related work and conclusion are presented in Sections 6 and 7, respectively.

Timeline (ms)	HRJN*		PRJ		cPRJ (Proposed)	
	Hotel RT: 500	Restaurant RT: 1000	Hotel RT: 500	Restaurant RT: 1000	Hotel RT: 500	Restaurant RT: 1000
500	C1	C1	C1	C1	C1	C1
1000	WAIT		C2		C2	C1
1500	C2	WAIT	C3		WAIT	C2
2000			C4			
2500	WAIT	C2	C5		C3	
3000	C3	WAIT	STOP	C3	WAIT	C3
3500	WAIT	C3		STOP	STOP	
4000	STOP	STOP				STOP
I/O COST: 3+3 = 6, TIME: 4000 ms		I/O COST: 5+3 = 8 TIME: 3000 ms		I/O COST: 3+3 =6, TIME: 3000 ms		

Fig. 1. Serial Data Access of HRJN* vs Parallel Data Access

2 Preliminaries

Consider a query Q whose answer requires accessing a set of Web services S_1, \dots, S_m , that can be wrapped to map their data in the form of tuples as in relational databases. Each tuple $t_i \in S_i$ is composed of an identifier, a join attribute, a score attribute and other named attributes. The tuples in every Web service are sorted in descending order of score, where the score reflects the relevance with respect to the query. Let $t_i^{(d)}$ denote a tuple at position d of S_i . Then $\sigma(t_i^{(d)}) \geq \sigma(t_i^{(d+1)})$, where $\sigma(t_i)$ is the score of the tuple t_i . Without loss of generality, we assume that the scores are normalized in the $[0,1]$ interval.

Each invocation to a Web service S_i retrieves a fixed number of tuples, referred to as chunk. Let (CS_i) denote the *chunk size*, i.e. the number of tuples in a chunk. The chunks belonging to a Web service are accessed in sequential order, i.e. the c -th chunk of a Web service will be accessed before $(c+1)$ -th chunk. Each chunk, in turn, contains tuples of S_i sorted in descending order of score. Furthermore, S_i provides one chunk of tuples in a specified time, which is referred to as its *average response time* (RT_i). Let $t = t_1 \bowtie t_2 \bowtie \dots \bowtie t_m$ denote a join result formed by combining the tuples retrieved from the Web services, where t_i is a tuple that belongs to the Web service S_i . This join result is assigned an aggregated score based on a monotone score aggregation function, $\sigma(t) = f(\sigma(t_1), \sigma(t_2), \dots, \sigma(t_m))$. The join results obtained by joining the data from these Web services are stored in a buffer S_{result} in descending order of their aggregate score.

2.1 Bounding Schemes

Let τ_i denotes the local threshold of a Web service S_i which represents an upper bound on the possible score of a join result that can be computed by joining any of the unseen tuples of S_i to either seen or unseen data of the rest of the Web services. The global threshold τ of all the Web services is the maximum among the local thresholds i.e. $\tau = \max\{\tau_1, \tau_2, \dots, \tau_m\}$.

The local threshold is updated with each data access to the corresponding Web service. Whereas, the global threshold is updated after every data access, independent of the accessed Web service. The bounding scheme is responsible

for computing τ , which represents an upper bound on the scores of possible join results, which can be formed by the unseen data. Thus, it helps in reporting the identified join results to the user. Let K denote the number of join results for which $\sigma(t) \geq \tau$, then these can be guaranteed to be the top- K . Figure 2(a) illustrates an example in which the global threshold is computed based on two possible bounding schemes based on the snapshot of execution, when all sources have fetched three tuples. The join predicate is the equality between the zip code attribute. These two bounding schemes *corner bound* and *tight bound* are further discussed below.

Corner Bound: The local threshold for a Web service S_i is calculated by considering the score of last seen tuple of S_i and maximum possible scores for the rest of the Web services. As an example, in Figure 2(a), the local threshold for S_1 is $\tau_1 = f(\sigma(t_1^{(3)}), \sigma(t_2^{(1)}), \sigma(t_3^{(1)})) = f(0.8, 1.0, 1.0) = 2.8$, assuming a simple linear score aggregation function. There is a drawback in using the threshold as computed by means of the *corner bound*. Indeed, it implicitly assumes that the first tuples of all the Web services formulate a valid join result, which may or may not be the case. Therefore, when more than two Web services are involved in a join, if the first tuples of all the Web services do not satisfy the join predicate, then the computed value of the *corner bound* threshold is not tight, in the sense that it might not be possible to from join results with unseen data that achieves that score. Note that HRJN* adopts a *corner bound* [7].

Tight Bound: It is possible to compute τ as a *tight bound* on the aggregate score of unseen join results [12]. The local threshold for a Web service S_i can be calculated by considering the score of the last seen tuple from S_i and the score of the partial join result, PJ_i , with maximum *possible* score which is formed by the rest of the Web services. Let $\mathcal{N}_i = \{i_1, \dots, i_n\}$ denote a subset of $\{1, \dots, m\}$ which does not contain the index i of Web service S_i , and $n = |\mathcal{N}_i|$, $0 \leq n < m$. There can be 2^{m-1} such distinct subsets. We find the join result with maximum possible score for each distinct subset \mathcal{N}_i^j , where $0 < j \leq 2^{m-1}$, and store it in $PJ(\mathcal{N}_i^j)$. The join results for a particular subset \mathcal{N}_i^j are computed by joining the seen tuples from the Web services whose indices are in \mathcal{N}_i^j and completing the join result with a tuple from the rest of the Web services i.e. $N - \mathcal{N}_i^j$, whose score is equal to the score of last seen tuple in the respective Web service. In this way, the join result in $PJ(\mathcal{N}_i^j)$ has one tuple from every Web service. The local threshold τ_i for S_i is computed as $\max(PJ(\mathcal{N}_i^j)), 0 < j \leq 2^{m-1}$. The maximum of all local thresholds is considered as global threshold. Further optimizations in the computation of the *tight bound* are discussed in [5]. When there are only two Web services [11], or the top scoring tuples in each service contribute to $PJ(\mathcal{N}_i^j)$, the *tight bound* and *corner bound* are equivalent. Figure 2(b) shows the average gain in terms of I/O cost and fetch time while using *tight bound* over *corner bound* using HRJN*, averaged over 10 different data sets.

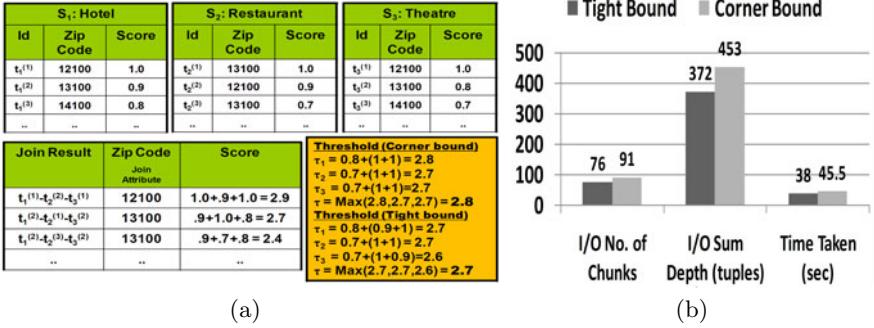


Fig. 2. (a) An example scenario, *tight* and *corner* bounding schemes. (b) Gain in I/O and time by using *tight bound* over *corner bound* with 4 Web services and $K=20$.

2.2 Data Pulling Strategy

The data pulling strategy provides a mechanism to choose the *most suitable* data source to be invoked at a given time during the execution [7]. The pulling strategy can be as simple as a round-robin strategy. HRJN* which focuses only on the optimization of the I/O cost, adopts a pulling strategy whereby the next service to be invoked is the one whose local threshold is equal to τ , the ties are broken by choosing the service which has extracted lesser number of tuples. The intuition of this pulling strategy is to keep all local thresholds as close as possible, which, due to monotonicity, is only possible by extracting the data from the data source with the highest local threshold. But the problem with this pulling strategy is that it takes longer time as shown in Figure 1. The objective of our work is to propose a pulling strategy that exploits the possibility of parallel access to the services. Such a strategy aims not only at minimizing the I/O cost, but also minimizing the time to fetch the data and hence, the time to report top- K join results. Our data pulling strategy is explained below in Section 3.1.1.

3 Methodology

3.1 Proposed Data Pulling Strategy

We stress on such a data pulling strategy which extracts data from all Web services in parallel. A naïve parallel pulling strategy, PRJ, keeps on extracting data from every data source till its respective local threshold becomes lesser or equal to the score of the *then* seen K -th join result. Figure 1 shows the comparison of different data pulling strategies. It shows that the I/O optimized HRJN* strategy has least I/O cost, but it takes more time to get top- K join results. Whereas, PRJ is only concerned with reducing the time to get top- K join results and it may result the extraction of unwanted data. This extraction of unwanted data is possible if a Web service stops well before the others, that is, its local threshold has reached below the score of the *then* top- K -th join result

in the output buffer S_{result} . In this case, there is a possibility that the other Web services having higher local thresholds produce join results with better aggregate score values, and terminate with an even higher local threshold. Resultantly, the Web service which stops earlier incurs extra data fetches. Therefore, in case of m Web services maximum $m - 1$ Web services may terminate earlier than the m -th Web service. Our proposed data pulling strategy extracts data from all the data sources in *controlled* parallel manner, the parallel data access helps minimizing the time to get top- K join results. Whereas, the I/O cost is minimized by pausing and resuming data extraction from the Web services. The pausing and resuming of data extraction from a Web service with lower local threshold, are performed on the basis of estimating the time to bring the local threshold of other Web services with higher local thresholds below or equal to its local threshold. This is explained in the Section 3.1.2. We use *tight* bounding scheme to compute the threshold values.

3.1.1 State Machine

In order to refrain from accessing the data that do not contribute to the top- K join results every Web service is controlled by using a state machine shown in Figure 3. The Web services are assigned a particular state after the completion of the processing of data fetched from any Web service. The *Ready* state means that the data extraction call should be made for this Web service. It is also the starting state for each Web service. A Web service S_i is put into *Wait* if we can fetch more data from any other Web service S_j and still its local threshold τ_j , will remain greater than or equal to τ_i . The *Stop* state means that further data extraction from this Web service will not contribute to determining the top- K join results. Lastly, the *Finish* state means that all the data from this Web service has been retrieved. The *Stop* and *Finish* states are the end states of the state machine. The difference between PRJ and the proposed cPRJ is that PRJ does not have *Wait* state, whereas, cPRJ controls the access to the unwanted data by putting the Web services into *Wait* state. On retrieving a chunk of tuples from Web service S_i the following operations are performed in order:

1. Its local threshold τ_i is updated and it is also checked if the global threshold τ also needs to be updated.
2. New join results are computed by joining the recently retrieved tuples from S_i with the tuples already retrieved from all other Web services.
3. All join results are stored in the buffer S_{result} in descending order of score. The size of the buffer S_{result} is bound by the value of K . All join results having aggregated score above τ are reported to the user.
4. The state for S_i is set using *setState* function shown in Figure 4(a). If S_i has extracted all its data then it is put to *Finish* state and τ_i is set to 0.

Apart from this the following operations are also performed:

1. Every Web service S_i , which is not in *Stop* or *Finish* state, is checked and is put into *Stop* state, if $\sigma(t_{result}^{(K)}) \geq \tau_i$.

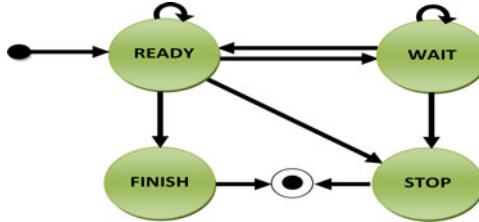


Fig. 3. The state machine according to which each Web service is manipulated

2. A Web service S_i that is in *Wait* state is put to *Ready* state, if there is no other Web service S_j which is in *Ready* state and τ_j is greater than τ_i , and S_j needs more than one chunk to bring τ_j lower than τ_i , and the minimum time needed to bring τ_j less than τ_i is greater than RT_i .

The state transitions are exemplified below in Section 3.1.3.

3.1.2 Time to Reach (ttr)

Data pulling strategy issues the data extraction calls by analyzing the local thresholds of the Web services. Particularly, the decisions to put a service from *Ready* to *Wait*, and *Wait* to *Ready* state are based on the computation of time to reach (*ttr*). Therefore, in order to clearly understand these state transitions we need to understand the computation of *ttr*. On completion of a data fetch from Web service S_i we identify all the Web services which are in *Ready* state and have higher local threshold value than τ_i , and put them in a set J . For each Web service S_j , in set J , we compute *time to reach*, (ttr_j), which is the time that S_j will take to bring τ_j below τ_i . The highest value of ttr_j is considered as *ttr* for Web service S_i . If *ttr* is greater than RT_i then S_i is put into *Wait* state, otherwise, it remains in *Ready* state.

The estimation of *ttr* involves the calculation of decay in score for the Web service S_j . We use Autoregressive Moving Average forecasting method [2] for the calculation of score decay. After estimating the unseen score values we can compute the total number of tuples needed to bring the τ_j lower than the value of τ_i . This number is then divided by the *chunk size* of S_j i.e. CS_j , to get the number of *chunks* to bring the threshold down. If number of *chunks* are less than one, i.e. the after getting the data from the currently extracted chunk τ_j will fall below τ_i , then ttr_j is set to 0. Otherwise, number of *chunks* are multiplied by RT_j , and the elapsed time ET_j , the time since the last data extraction call is issued for S_j is subtracted i.e. $ttr_j = (\text{chunks} \times RT_j) - ET_j$.

3.1.3 State Transitions in the State Machine

The state transitions shown in Figure 3 are exemplified below with the help of Figure 4(a). There are 3 Web services S_1, S_2 and S_3 with $RT_1 = 400ms$, $RT_2 = 700ms$ and $RT_3 = 900ms$, for simplicity, score decay for all Web services is kept linear.

Ready to Finish: If a Web service has been completely exhausted, i.e. all the data from it has been retrieved then its state is changed from *Ready* to *Finish*. A

Web service can be put to *Finish* state only when it is in *Ready* state and makes a data extraction. Figure 4(a) shows that after 2800ms, S_2 is put from *Ready* to *Finish* state. **Ready to Stop and Wait to Stop:** If a Web service is in *Ready*

or *Wait* states then it should be put into *Stop* state if the following condition holds: if S_{result} already holds K join results, then the algorithm compares the local threshold τ_i with $\sigma(t_{result}^{(K)})$, the score of $K - th$ join result in S_{result} . If τ_i is less than or equal to it then it assigns *Stop* state to S_i . This essentially means that further extraction of data from this Web service will not produce any join result whose score is greater than the join results already in S_{result} . Figure 4(a) shows that after 2100ms the Web service S_3 is put from *Wait* to *Stop* state as its τ_3 is lower than $\sigma(t_{result}^{(K)})$. Whereas, S_1 is put from *Ready* to *Stop* state at 2500ms.

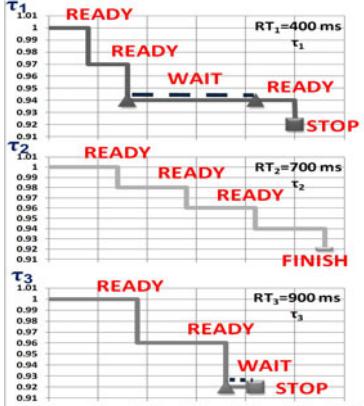
Ready to Ready, Ready to Wait, Wait to Ready and Wait to Wait: A Web service in *Ready* state is put to *Wait* state, or a Web service in *Wait* state is put to *Ready* state by analyzing the local thresholds of all other Web services which are in *Ready* state. Figure 4(b) presents the algorithm for *setState* function. Below is the explanation of the algorithm for a Web service S_i :

- Consider a set J containing all the Web services having local thresholds greater than that of τ_i and are in *Ready* state. The algorithm estimates the *time to reach* (ttr_j), for all Web services $S_j \in J$ to bring τ_j lower than τ_i as explained in Section 3.1.2.
- Thus, ttr_j is computed for all Web services in J and the maximum of these values is retained as ttr .
- If S_i is in *Ready* or *Wait* state and ttr is greater than or equal to RT_i then S_i is assigned *Wait* state, otherwise, it is put to *Ready* state.

Figure 4(a) shows that, after 800ms, S_1 is put from *Ready* to *Wait* state because of bootstrapping phase, as no more than 2 data extraction calls are allowed during this phase from any Web service. This is explained below in this section. However, even after finishing the bootstrapping, at 900ms, it remains in *Wait* state as ttr_2 is 1900ms which is greater than RT_1 . S_1 continues to be in *Wait* state at 1400ms and at 1800ms, as ttr is greater than RT_1 . Similarly, at 1800ms, S_3 is put to *Wait* state from *Ready* state, as ttr_2 is 1700ms.

After 2100ms S_1 is put from *Wait* to *Ready* state as at this time ttr_2 is 0. Therefore, we need to resume data extraction from S_1 as well. Lastly, S_2 remains in *Ready* state during all the state transitions, till it moves from *Ready* to *Finish* state at 2800ms because it remains the Web service with highest local threshold i.e $\tau_2 = \tau$.

Bootstrapping: At the beginning data is extracted from all Web services in parallel. The phase before extraction of at least one chunk from all Web services is considered as bootstrapping phase. The Web services with smaller response time may fetch too much data in this phase. So, during bootstrapping, we limit maximum two data fetches from a particular Web service. The rationale is that these Web services have much shorter response time so they can catch



(a)

```

Function setState (SrcArray, Si, Sresult)
Inputs: SrcArray: containing all the Web services;
          Si: the source under consideration;
          Sresult: all join results found so far.
Output: state of source 'R' READY, 'W' WAIT, 'S' STOP
1. if(Sresult.size ≥ K) then
2.   if(tresult(k).score ≥ τi) then
3.     return 'S'
4. ttr ← 0
5. for j ← 0 to SrcArray.length
6.   if Sj = Si OR Sj.State ≠ 'R' then continue
7.   diff ← τj - τi
8.   ttrj ← 0
9.   if (diff > 0) then
10.    ttrj ← computeTimeToReach(Sj, diff)
11.   if(ttrj > ttr) then ttr ← ttrj
12. if(ttr ≥ RTi) then
13.   return 'W'
14. return 'R'

```

(b)

Fig. 4. (a) Execution of cPRJ with 3 Web services, over timeline against local thresholds. (b) The setState algorithm

up the other Web services with higher response times. It can be observed in Figure 4(a) that S_1 is put to *Wait* state after making two fetches, at 800ms. Similarly, at 400ms S_1 , and at 700ms S_2 , are allowed to perform second fetch. The bootstrapping phase ends after 900ms.

Adaptivity to the Change in RT: Sometimes it is possible that a Web service S_i does not demonstrate the same response time as anticipated. To determine this, the proposed algorithm always computes the response time for every chunk and computes average of the last 3 observed response times. If the deviation is within 10% of the existing response time value, then the latter is retained. Otherwise, RT_i is assigned the average of its last 3 observed RT values.

4 Concurrent Pre-fetching with cPRJ

It is possible to profile a Web service S_i and identify if more than one concurrent calls can be issued to it. In such cases, instead of fetching one chunk at a time from S_i , the algorithm might issue $S_{i(conc)}$ concurrent calls. This helps in speeding up data fetching even further, as it acquires data from $S_{i(conc)}$ chunks in RT_i , the same time in which the baseline cPRJ gets one chunk.

This also requires modifications in the *setState* function while calculating the *ttr*, by incorporating the number of concurrent chunks extracted by S_i . Also, while issuing the data extraction calls, the algorithm has to check the number of chunks a Web service needs to bring its local threshold down to $\sigma(t_{result}^{(K)})$. If they are greater than or equal to $S_{i(conc)}$ then all concurrent data extraction calls can be issued. Otherwise, the number of calls is that suggested by the calculation.

This variant certainly reduces the time to find the top- K join as compared to the baseline version of cPRJ. However, it may incur some additional I/O cost because of concurrent data extraction.

Concurrent accesses to a Web service might also be considered an ethical issue as it prevents the other users from accessing the same service at the same time, especially in peak hours. However, in our case the total number of calls to a Web service will still remain almost the same even if we issue them concurrently. Secondly, the number of concurrent calls, in general is not high, and it should be issued only for the Web services with larger response times, or which exhibit a very low decay in their scores. As an example, in case of extracting data from the Web services `venere.com` and `eatinparis.com`, shown in Table 1, it will be useful to extract the concurrent chunks from them according to the ratio between their response times, provided their score decay per chunk is observed to be in the same ratio.

5 Experimental Study and Discussion

5.1 Methodology

Data Sets: We have conducted the experiments on both synthetic data, and real Web services. The experiments are based on the query in Example 1 by generating many different synthetic data sources with various parameter settings. The relevant parameters are presented in Table 2. The real Web services used for the experiments are presented in Table 1. These real services were queried for finding the best combination of hotels and restaurants in a city, for many different cities. For each city, we find the best combination of hotels and restaurants located in the same zip code. In order to consider more than two Web services, we have also extracted information about museums and parks from the real Web services. The experiments with synthetic data are performed with diverse and homogeneous settings of values for the parameters in Table 2. Homogeneous settings help us understanding the behaviour of individual parameter whereas, diverse settings help us simulating the real environment Web services, as we have observed that most of them have diverse parameter settings. For fairness, we compute these metrics over 10 different data sets and report the average. The experiments with the real Web services are conducted by fetching the data from real Web services for 5 different cities and the averaged results are reported.

Table 1. Real Web services used for experiments

Web Services	Type of Information	Response Time	Chunk Size
1 www.venere.com	Hotels	900 ms	15
2 www.eatinparis.com	Restaurant (only for Paris)	350 ms	6
3 Yahoo! Local	Hotels, Restaurants, Museums, Parks	800-1200 ms	10
4 www.yelp.com	Hotels, Restaurants, Museums, Parks	900-1100 ms	10

Table 2. Operating Parameters (defaults in bold)

Full Name	Parameter	Tested Values
Number of results	<i>K</i>	1,20,50,100
Join Selectivity	<i>JS</i>	0.005, 0.01, 0.015 , 0.02
Score Distribution	<i>SD</i>	Uniform Distrib., Zipfian Distrib. , Linear Distrib., Mixed
Response Time	<i>RT</i>	500/500 , 500/1000, 500/1500
Chunk Size	<i>CS</i>	5/5 , 5/10, 5/15
Number of relations	<i>m</i>	2,3,4

Approaches: We compare three algorithms, HRJN*, PRJ and the proposed cPRJ while using *tight* bounding scheme. An important consideration is that HRJN* augmented with *tight* bounding cannot be beaten in terms of I/O cost, whereas PRJ cannot be out-performed in terms of time taken, provided the time taken for joining the data is negligible. Therefore, the proposed algorithm, cPRJ carves out a solution that deals in the trade off between I/O cost and time taken. Indeed, the parallel approaches should be efficient in terms of time taken than the serial data accessing HRJN* approach yet, the purpose of including HRJN* in the comparison is to elaborate the gain in terms of I/O cost when using cPRJ instead of PRJ.

Evaluation Metrics: The major objective of the proposed approach is to reduce the time taken to get the top- K results by minimizing the data acquisition time with the help of parallelism. So, we consider *time taken* as the primary metric for comparing different algorithms. This is the wall clock time, that is, starting from the first fetch till the K -th join result is reported. The reduction in time is obtained by compromising on possibly some extra data extraction as compared to HRJN*. Therefore, we consider *sum depths* [5], total number of tuples retrieved from all Web services, as other metric for comparing the different algorithms.

5.2 Results

Experiments with Synthetic Data: In Figure 5 we show the results of the experiments for *CS*, *RT* and *SD* parameters while joining two Web services. In case of the homogeneous setting of the parameters, i.e. keeping all the parameters to the default values and setting different values for one of the three above mentioned parameters. This results into termination of data extraction from $(m - 1)$, in this case, one data source earlier than the other data source, as explained in section 3.1. The proposed cPRJ algorithm is also based on these three parameters. Figure 5(b) shows that cPRJ incurs 1% more and PRJ incurs 8% more I/O cost than HRJN* in case of different *CS* values. For different values of *RT* and *SD* both HRJN* and cPRJ take the same I/O cost, and PRJ takes 8% more and 10% more I/O cost than HRJN* for different values of *RT* and *SD*, respectively. If we augment all these in one scenario then cPRJ incurs 3% more I/O cost than HRJN* and PRJ costs 29% more I/O cost than HRJN*.

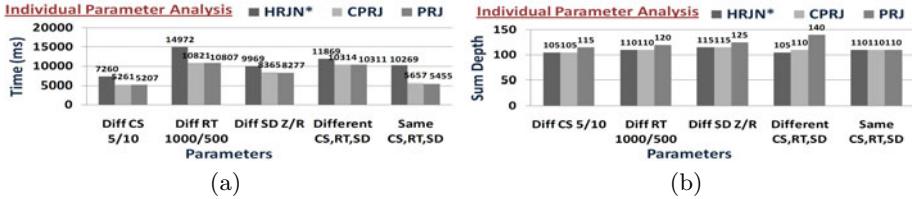


Fig. 5. Performance comparison of the algorithms on synthetic data sources for the parameters shown in Table 2

Whereas, Figure 5(a) shows that for all cases the time taken by both parallel approaches is almost same and is much lower than HRJN*. However, if CS , RT and SD are identical for all data sources, then all three approaches have almost same I/O cost and both parallel approaches take same time.

The overall performance of cPRJ is much better than PRJ in case of diverse parameter settings, as it has almost same I/O cost as of HRJN* whereas, it takes almost same time as of PRJ, whereas, PRJ has higher I/O cost than HRJN*. Thus, in the diverse settings it brings the best of both worlds.

Real Web Services: The experiments with the real Web services, which in general, have diverse parameter settings, confirm the same observations made on synthetic data, i.e. overall cPRJ performs much better than PRJ. We performed experiments for the query in Example 1 while interacting with the real Web services to get top- K join results. We have used different Web services, presented in Table 1. Figure 6(a) shows that both parallel approaches take same amount of time which is 20-25% less than HRJN*. The difference in time increase by increasing K . Figure 6(b) shows that the I/O cost incurred by proposed cPRJ is 5% more than ideal HRJN*, whereas, PRJ takes 8-10% extra data fetches. We have also performed experiments by varying the number of Web services involved in the search query. We add data for museums as third and data for parks as fourth Web service in our search. We use [Yahoo! Local](#) and [yelp.com](#) to fetch data for museums and parks. The results shown in Figure 6(c) show that both parallel approaches take almost same time and this time is 14-35% less than HRJN*. The difference in time taken by parallel approaches and HRJN* increases by adding more data sources, i.e., by increasing the value of m . The results presented in 6(d) demonstrate that cPRJ takes 4-11% more I/O cost than HRJN*, whereas, PRJ takes 13-38% more I/O cost than HRJN*.

The experimental results also show that other three parameters JS , m and K do not have any impact *alone*. They cannot be responsible for the early termination of a *single* data source. However, if SD , RT and CS have heterogeneous values, and if the overall impact of these values is that they enforce one or more data sources to terminate earlier than the others while using the parallel approaches, then JS , m and K also come into play. The results shown in Figures 6(a) and 6(b) show the role of K and Figures 6(c) and 6(d) show the behaviour of number of data sources m , involved in a query.

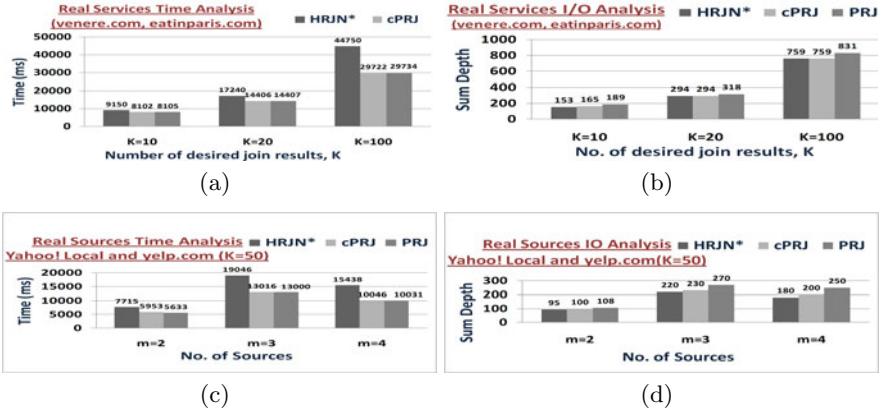


Fig. 6. Performance of the algorithms with real services. Figures (a) and (b) are for the experiments with [venere.com](#) and [eatinparis.com](#). Figures (c) and (d) are experiments with different number of sources using [Yahoo! Local](#) and [yelp.com](#)

The method used to compute *ttr* is supposed to provide accurate estimates when the score decay is smooth. When this is not the case (e.g. when ranking of hotels is induced by the number of stars), it tends to underestimate the score decay. If it underestimates the score decay then the state machine may pause a Web service unnecessarily, which may increase the overall time. Conversely, in case of overestimation of the score decay, the state machine may not pause a Web service at right time, hence, it may incur extra I/O cost.

Concurrent Pre-fetching: The results in Figure 7 are based on an experiment which issues different number of concurrent calls to the real Web services, [venere.com](#) having response time 900ms and [eatinparis.com](#) having response time 350ms. We issue concurrent calls in two ways, firstly, based on the ratio between the response times of the two sources, and secondly, we issue three concurrent calls for both data sources without any consideration. The results show that in both cases the time decreases by almost 62% of the baseline cPRJ approach. This implies that [venere.com](#) takes most of the time to fetch the data to produce required number of join results, whereas, [eatinparis.com](#)

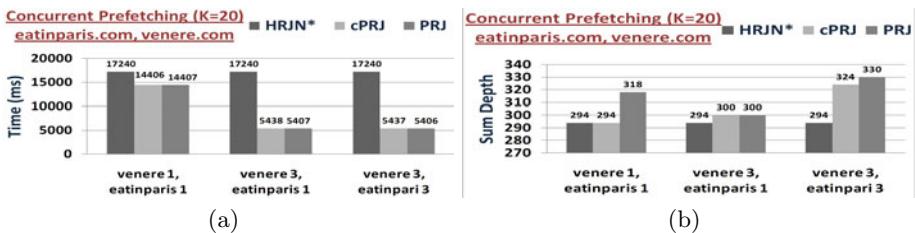


Fig. 7. Figures (a) and (b) show the comparison of time and I/O for $K=20$, where cPRJ and PRJ perform different number of concurrent fetches on real Web services

takes one third or lesser time to fetch its data from the same purpose. Therefore, when we fetch three concurrent chunks from `venere.com` and one chunk from `eatinparis.com`, we get the best result. While observing the difference in the I/O cost, we find that first method of concurrent calls has proven to be almost as effective as baseline cPRJ whereas the second one has incurred 10% extra I/O cost than the baseline cPRJ. More than one concurrent data fetches from a Web service certainly minimize the time, however, using it in a *smarter* fashion can also help avoiding possible extra I/O cost.

6 Related Work

We are considering rank join operators with only sorted access to the data sources, therefore, we only discuss the existing solutions while respecting this constraint. The NRA algorithm [4] finds the top- K answers by exploiting only sorted accesses to the data. This algorithm may not report the exact object scores, as it finds the top- K results using bounds; score lower bound and score upper bound; computed over their exact scores.

Another example of no random access top- K algorithms is the J* algorithm [1]. It uses a priority queue containing partial and complete join results, sorted on the upper bounds of their aggregate scores. At each step, the algorithm tries to complete the join combination at the top of the queue selecting the next input stream to join with the partial join result and reports it as soon as it is completed. This algorithm is expensive in terms of memory and I/O costs as compared to HRJN* in most of the cases.

HRJN [7] is based on symmetrical hash join. The operator maintains a hash table for each relation involved in the join process, and a priority queue to buffer the join results in the order of their scores. The hash tables hold input tuples seen so far and are used to compute the valid join results. It also maintains a threshold τ and uses a data pulling strategy to compute join results. Some recent improvements in HRJN algorithm are presented in [5] and [12]. These algorithms use *tight bound* to compute top- K join results and show their comparative analysis.

Another interesting and objectively similar work has been done in [10], but the proposed algorithm *Upper* incorporates both serial and random accesses to the data sources, whereas, in our case we only use sorted access to the data sources. The commonality between the two approaches is that both cPRJ and *Upper* minimize the data extraction time by issuing concurrent data extraction calls and also exploit the pre-fetching of data while respecting the number of maximum concurrent fetches to the data sources.

7 Conclusion

We have proposed a new rank join algorithm cPRJ, for multi-way rank join while using parallel data access. This algorithm is specifically designed for distributed data sources which have a non-negligible response time e.g. the Web services available on the Internet. It uses a score guided data pulling strategy

which helps computing the top- K join results. The results based on the experiments conducted on synthetic and real Web services show that the I/O cost of the proposed approach is nearly as low as optimal I/O cost of HRJN*, and it computes the join results as quick as PRJ approach which cannot be beaten in terms of time taken. cPRJ exhibits its strengths when the Web services have such diverse parameter settings which enforce one or more data sources to terminate earlier than any other data source while accessing them in parallel. We have also exploited the concurrent data fetching property of the Web services in order to get the data in even quick time. This reduces the time to compute the joins even further, but at higher I/O cost than baseline cPRJ. As a next step, we anticipate that this parallel rank join operator can be enhanced for pipe joins.

Acknowledgments

This research is part of the “Search Computing” project, funded by the European Research Council, under the 2008 Call for “IDEAS Advanced Grants”.

References

1. Nastev, A., Chang, Y., Smith, J.R., Li, C., Vittor, J.S.: Supporting incremental join queries on ranked inputs. In: VLDB Conference
2. Brockwell, P.J.: Encyclopedia of Quantitative Finance (2010)
3. Ceri, S., Brambilla, M. (eds.): Search Computing II. LNCS, vol. 6585. Springer, Heidelberg (2011)
4. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences* 66(4), 614–656 (2003)
5. Finger, J., Polyzotis, N.: Robust and efficient algorithms for rank join evaluation. In: SIGMOD Conference, pp. 415–428 (2009)
6. Guntzer, U., Balke, W., Kiessling, W.: Towards efficient multi-feature queries in heterogeneous environments. In: International Conference on Information Technology: Coding and Computing, Proceedings, pp. 622–628 (2001)
7. Ilyas, I., Aref, W., Elmagarmid, A.: Supporting top-k join queries in relational databases. *The VLDB Journal* 13(3), 207–221 (2004)
8. Ilyas, I., Beskales, G., Soliman, M.: A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys* 40(4), 1 (2008)
9. Mamoulis, N., Theodoridis, Y., Papadias, D.: Spatial joins: Algorithms, cost models and optimization techniques. In: *Spatial Databases*, pp. 155–184 (2005)
10. Marian, A., Bruno, N., Gravano, L.: Evaluating top- k queries over web-accessible databases. *ACM Trans. Database Syst.* 29(2), 319–362 (2004)
11. Martinenghi, D., Tagliasacchi, M.: Proximity rank join. In: PVLDB, vol. 3(1), pp. 352–363 (2010)
12. Schnaitter, K., Polyzotis, N.: Optimal algorithms for evaluating rank joins in database systems. *ACM Trans. Database Syst.* 35(1) (2010)

Assessing Fault Occurrence Likelihood for Service-Oriented Systems

Amal Alhosban¹, Khayyam Hashmi¹, Zaki Malik¹, and Brahim Medjahed²

¹ Department of Computer Science

Wayne State University, MI 48202

{ahusban,khayyam,zaki}@wayne.edu

² Department of Computer & Information Science

The University of Michigan - Dearborn, MI 48128

brahim@umich.edu

Abstract. Automated identification and recovery of faults are important and challenging issues for service-oriented systems. The process requires monitoring the system's behavior, determining when and why faults occur, and then applying fault prevention/recovery mechanisms to minimize the impact and/or recover from these faults. In this paper, we introduce an approach (defined FOLT) to automate the fault identification process in services-based systems. FOLT calculates the likelihood of fault occurrence at component services' invocation points, using the component's past history, reputation, the time it was invoked, and its relative weight. Experiment results indicate the applicability of our approach.

Keywords: Service-oriented architecture, Fault tolerance, Reliability.

1 Introduction

Over the past decade, we have witnessed a significant growth of software functionality that is packaged using standardized protocols either over Intranets or through the Internet. System architectures adhering to this development approach are commonly referred to as service-oriented architectures (SOA). In essence, SOAs are distributed systems consisting of diverse and discrete software services that work together to perform the required tasks. Reliability of an SOA is thus directly related to the component services' behavior, and sub-optimal performance of any of the components degrades the SOA's overall quality. Services involved in an SOA often do not operate under a single processing environment and need to communicate using different protocols over a network. Under such conditions, designing a fault management system that is both efficient and extensible is a challenging task. The problem is exacerbated due to security, privacy, trust, etc. concerns, since the component services may not share information about their execution. This lack of information translates into traditional fault management tools and techniques not being fully equipped to monitor, analyze, and resolve faults in SOAs.

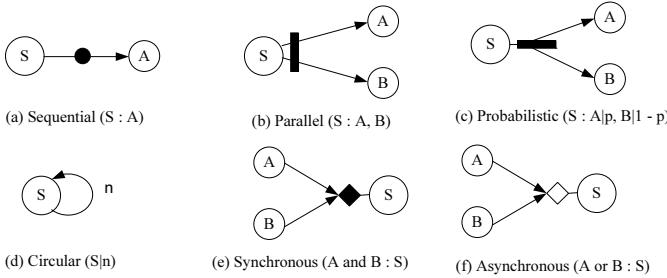
In this paper, we present a fault management approach (Fault Occurrence Likelihood esTimation: FOLT) for SOAs. We assume that component services do not share their execution details with the invoking service (defined as an orchestrator). The orchestrator only has information regarding the services' invocation times and some other *observable* quality of service (QoS) characteristics. We propose to create fault expectation points in a SOA's invocation sequence of component services to assess the likelihood of fault occurrence. Fault recovery plans are then created for these expectation points and are stored in a data repository to be retrieved and executed when the system encounters a fault at runtime. Due to space restrictions we only focus on the former in this paper, i.e., assessing the likelihood of a fault's occurrence. The latter, i.e., "fault recovery" requires independent discussion.

The paper is organized as follows. Section 2 presents an overview of the service-oriented architecture. We then discuss service invocation models used there in, and overview the relationship between services in each model, and the expected faults for each invocation model. The fault assessment techniques are discussed in Section 3. We present some experiments and analysis of FOLT in Section 4, while Section 5 provides an overview of related work. Section 6 concludes the paper.

2 Service-Oriented Architecture

In this section, we present a brief overview of service-oriented architectures and the different invocation models used by the composition orchestrators. SOA is defined as "a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains" [11]. Boundaries of SOAs are thus *explicit*, i.e., the services need to communicate across boundaries of different geographical zones, ownerships, trust domains, and operating environments. Thus, explicit message passing is applied in SOAs instead of implicit method invocations. The services in SOAs are *autonomous*, i.e., they are independently deployed, the topology is *dynamic*, i.e., new services may be introduced without prior acknowledgment, and the services involved can leave the system or fail without notification. Services in SOAs *share* schemas and contracts. The message passing structures are specified by the schema, while message-exchange behaviors are specified by the contracts. Service compatibility is thus determined based on explicit policy definitions that define service capabilities and requirements [12].

Two major entities are involved in any SOA transaction: Service consumers, and Service providers. As the name implies, service providers provide a service on the network with the corresponding service description [8]. A service consumer needs to discover a matching service to perform a desired task among all the services published by different providers. The consumer binds to the newly discovered service(s) for execution, where input parameters are sent to the service provider and output is returned to the consumer. In situations where a single service does not suffice, multiple services could be *composed* to deliver the required functionality [11].

**Fig. 1.** Major SOA Invocation Models

Each service in an SOA may be invoked using a different invocation model. Here, an invocation refers to *triggering a service (by calling the desired function and providing inputs) and receiving the response (return values if any) from the triggered service*. An SOA may thus be categorized as a ‘composite service’, which is a conglomeration of services with invocation relations between them. There are six major invocation relations (see Figure 1). (a) Sequential Invocation: the services are invoked in a sequence, (b) Parallel Invocation: multiple services are invoked at the same time, (c) Probabilistic Invocation: one service is invoked from the multiple options, (d) Circular Invocation: a service invokes “itself” x times, (e) Synchronous Activation: all services that were invoked need to complete before the composition can proceed, and (f) Asynchronous Activation: completion of one of the invoked services is enough for the composition to proceed [4]. Due to space restrictions, the detailed discussion about these models is omitted here. The interested reader is referred to [9].

3 Fault Occurrence Likelihood

In this section, we present our approach Fault Occurrence Likelihood esTimation (FOLT) which estimates the likelihood of fault for component services. For the sake of discussion, each service is treated as an independent and autonomous component. This component either performs its desired behavior (i.e., success) or fails to deliver the promised functionality (i.e., fault). FOLT depends on three major factors: the service’s past fault history, the time it takes to complete the required task in relation to the composition’s total execution time, and the service’s weight (i.e., importance) in the composition (in relation to other services invoked). Since, a composed service using one or more of the invocation models described above, may encounter a fault during its execution, the likelihood of encountering a fault is directly proportional to the system’s complexity, i.e., the more the invocation models involved, the greater the likelihood of a fault’s occurrence. FOLT output values are thus influenced by the invocation model(s) used in the composition. In other words, fault occurrence likelihood is different from one invocation model to the other. In the following, we provide

details of the FOLT approach. We first provide an illustrative scenario where FOLT is applied, and then detail the proposed approach's architecture, and technique.

3.1 Sample Scenario

A student (Sam) intends to attend a conference in London, UK. He needs to purchase an airline **ticket** and reserve a **hotel** for this travel. Moreover, he needs some **transportation** to go from the airport to the hotel and from the hotel to other venues (since this is the first time he's visited the UK, he intends to do some "Site-seeing" also). Sam has a restricted budget, so he is looking for a "deal".

Assume that Sam would be using a SOA-based online service (let's call it *SURETY*) that is a one-stop shop providing all the five options (airline ticket, hotel, attractions, transportation and discounts) through outsourcing. *SURETY* provides many services such as: attraction service which outsources to three services (representing individual services): Art, Museums, and Area tours. This service provides arrangement to visit different areas through sub-contractor companies. For clarity, Figure 2 shows the options at one level. Sam may select Art, Museum, Area tours, or any combination of these services. In terms of transport options, Sam can either use a taxi service, or move around in a rental car, bus, or bike. The different transport companies provide services based on the distance between the places (attractions, etc.) Sam plans to visit. *SURETY* also provides a package optimization service that finds "deals" for the options chosen by Sam.

In Figure 2, the potential services are shown for clarity from "Get request" (when *SURETY* receives Sam's request) to "Send result" states (when *SURETY* sends result(s) to Sam). This is done to show a combination of different invocation models. In reality, service invocations may not follow such a *flat* structure. Since Sam is looking for a travel arrangement that include: booking a ticket, booking a hotel, transportation (rental car, bike or bus) or taxi, and visiting some places, some of these services can be invoked in parallel (here we assume that *SURETY* provides such an option). Booking a ticket and finding attractions is an example of *parallel invocation*. Among the three choices that Sam can select from (Area tours, Museums, and Art), for area attractions, he has to make a choice among these service instances; this is an example of *probabilistic invocation*. Similarly, taxi or rental car, bike and bus services can be classified as probabilistic invocations since *SURETY* has to invoke one service from among multiple services. *SURETY* then provides the results of transport selection to the Package Optimization service, which hunts for available discounts (e.g., if the customer uses the system for more than one year he will get a 20%, etc.). This invocation is an example of *asynchronous invocation*, as one of the transport selections will suffice. *SURETY* then sends the final selection itinerary to Sam. In the following, we show how to use these invocation points to assess the likelihood of a fault's occurrence (similar to [2]).

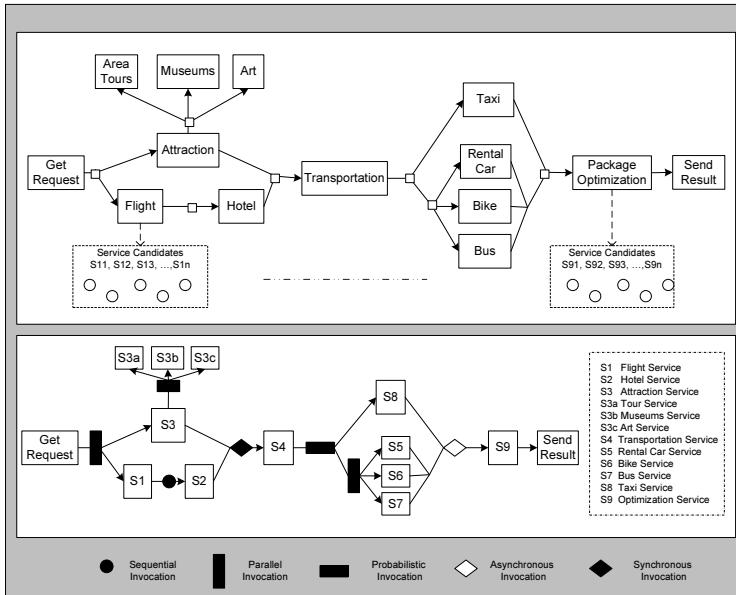


Fig. 2. Scenario with Invocation Models

3.2 Proposed Architecture

In this section, we discuss the architecture of FOLT. FOLT is divided into three phases. In Phase 1, we assess the fault likelihood of the service using different techniques (HMM, Reputation, Clustering). In Phase 2, we build a recovery plan to execute in case of fault(s). Finally, in Phase 3, we calculate the overall system reliability based on the fault occurrence likelihoods assessed for all the services that are part of the current composition. In this paper, we present only the work related to Phase 1. Phase 2 and 3 require independent discussion, which are not presented here due to space restrictions. Details of Phase 1 follow.

3.3 Phase 1: Fault Occurrence Likelihood Assessment

In this Phase, we calculate the fault occurrence likelihood for the service to assess its reliability. The notations used hereafter are listed in Table 1. Most of the terms in the table are self-explanatory. Brief descriptions of other symbols follow: λ_i is the ratio of the time taken by $service_i$ (to complete its execution), to the total composition execution time. On the other hand, λ'_i is the ratio of the time taken by $service_i$ to the total time “remaining” in the composition, from the point when $service_i$ was invoked. Δ_i is the first-hand experience of an invoking service regarding a component $service_i$ ’s propensity to fault. For cases where the invoker has no historical knowledge of $service_i$ (i.e., the two services had no prior interaction), $\Delta_i = 0$. Similarly, Δ'_i is the second-hand experience regarding a $service_i$ ’s faulty behavior. This information is retrieved

Table 1. Definition of Symbols

Symbol	Definition
T	The total execution time.
t_0	Start time.
t_n	End time.
t_i	Time at which a new service is invoked.
k	Number of services.
$P(x)^t$	Fault occurrence likelihood for service _x when invoked at time t .
λ_i	Weight of service _i in relation to T .
λ'_i	Weight of service _i in relation to $(T - t_i)$.
Δ_i	First-hand fault history ratio of service _i .
Δ'_i	Second-hand fault history ratio of service _i .
$f(s_i)$	The priority of service _i in the composition.

from other services that have invoked $service_i$ in the past. We assume that trust mechanisms (such as [8]) are in place to retrieve and filter service feedbacks. $f(s_i)$ is the assigned weight of a $service_i$ in the whole composition. It provides a measure for the importance of $service_i$ in relation to other component services invoked, where $\sum_{i=1}^n f(s_i) = 1$.

FOLT architecture (Figure 3) is composed of several modules. These are, *History Module*: This module keeps track of an individual service's propensity to fault. The information is stored in a *History Repository* that includes the service name, invocation time, reported faults (if any), and a numerical score. The *Estimation Module* calculates the fault occurrence likelihood for a service in a given context (execution history). An optional *Priority Module* is used sometimes (details to follow) to indicate the service priority assignment by the invoker in a given execution scenario. Lastly, the *Planning Module* creates plans to recover from encountered faults, and prevent any future ones. As mentioned earlier, details of the module are not the focus of this work.

In summary, the designers store some of the plan details in a plan repository while others are generated at run time. Each plan contains specific fields such as: Plan ID, Plan Name, Plan Duration time, Plan Steps and Plan Counter. When FOLT decides to generate a plan, the system starts the dynamic generation process. The generated plan depends on the chosen invocation model. When the orchestrator invokes a service at any given time (invocation point), it calculates the fault history ratio for the invoked service. Here, we use the maximum value among the external ratio (service's second-hand experience as observed by the community) and internal ratio (first-hand experience of the orchestrator). The system then calculates the fault occurrence likelihood of the invoked service. If the likelihood is greater than a pre-defined threshold (θ_1) the system builds a fault prevention plan. Otherwise, the system re-calculates the likelihood taking into consideration the priority of the current service and compares the value again with θ_1 . The purpose of this step is that non-critical services have no plans built for them, and the system can complete the execution even if a fault occurs in any of these services. The newly created plan is tested using a series

of verifications. If the plan fails any of the tests, the system returns back to the planning module, and a new plan is created/checked. The process repeats for x number of times until a valid plan is found. If no plan is still found, the invoker/user is informed. Once a valid plan is created, it is stored in the repository. Then, If the likelihood is greater than another pre-defined threshold (θ_2) the system can execute this fault prevention plan.

Phase 1 is divided into multiple steps: calculating the service's weight (λ), calculating the time weight (λ'), calculating the internal history value (Δ_i) using a Hidden Markov Model, and calculating the external history value (Δ'_i) using clustering and reputation. The likelihood of a fault occurring at time t is defined by studying the relationship between the service's importance, time it takes to execute, and its past performance in the composition. Thus, each invocation model will have a different fault likelihood value. As mentioned earlier, λ is the ratio of the time that is needed to complete the service execution, divided by the total time of completing the execution of the whole system. Similar to the approach used in [10], we use this value of λ as one of the basic constructs

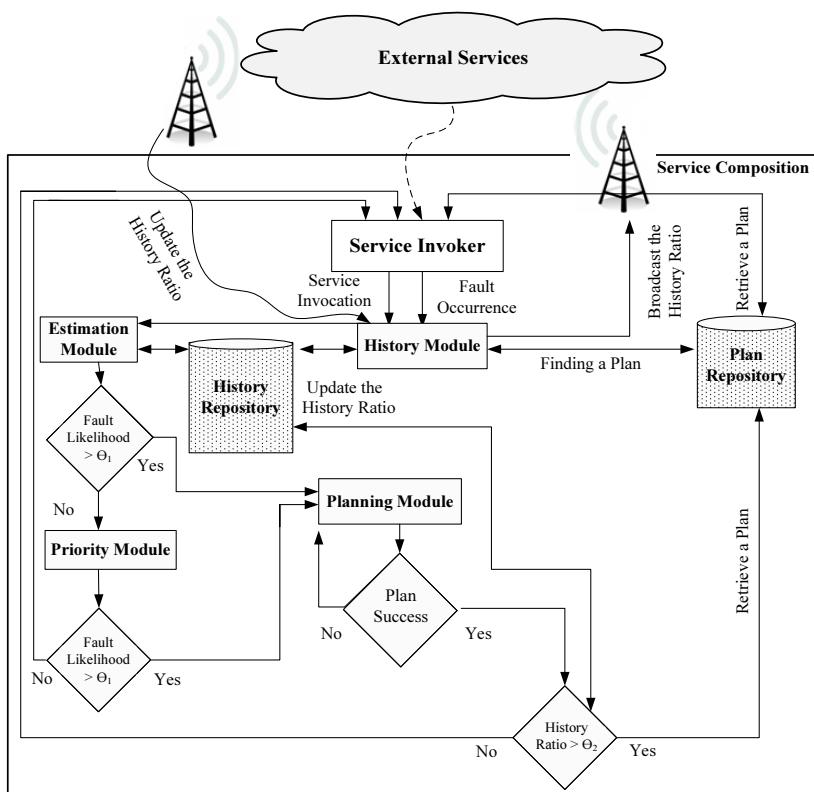


Fig. 3. FOLT Architecture

in FOLT to measure the (relative) weight of the invoked service to the rest of system time. The basic premise is that the likelihood of a fault occurrence for a long running service will be more than a service with very short execution time. Determining the service execution time could be accomplished in two ways. If the system does not know the execution time for a service, then the service's advertised execution time is used. On the other hand, after attaining experience with the service (prior invocations), the service execution time could be recorded and stored in the repository. Then:

$$\lambda_i = \frac{T(s_i)}{T} \quad (1)$$

$$\lambda'_i = \frac{T(s_i)}{T - t_i} \quad (2)$$

where λ_i is as described above, $T(s_i)$ is the total execution time of $service_i$, while t_i is the invocation time of $service_i$ (i.e., when the service was invoked).

A service's past behavior is assessed according to first-hand experience of the invoking service and second-hand experiences of other services obtained in the form of ratings via the community. These *experiences* are evaluated as a ratio of the number of times the service failed, divided by the total number of times the service was invoked. To assess the First-hand Experience, we use a Hidden Markov Model (HMM). The HMM provides the probability that the service will fail in the next invocation, based on the previous behavior of the service within the system. HMMs have proven useful in numerous research areas for modeling dynamic systems [7]. An HMM is a process with a set of states, set of hidden behavior and a transition matrix. In our architecture, all services stay in one of the two states: Healthy or Faulty (Figure 4).

Each time the composition orchestrator invokes service, it records the state of that service (Faulty or Healthy) along with the time of invocation. Let the vector V = the service behavior profile, then to asses the probability that $Service_i$ will be in the Faulty state in the next time instance:

$$P(Faulty|V) = P(Faulty|Healthy) + P(Faulty|Faulty) \quad (3)$$

FOLT also uses other services' experiences with $Service_i$ to assess its reliability. Services are divided into clusters based on their similarity (such as in [1]). These group of services are consulted for the reputation of $Service_i$. We assume

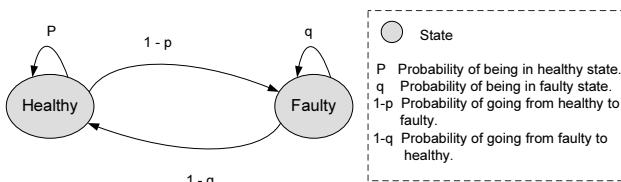


Fig. 4. Finite state machine for a HMM of the service

that other services are willing to share their reputation ratings, which are assimilated using our previous work in [8]. A combination of service time weights and service history ratios (using HMM, and reputation) is used to assess the fault occurrence likelihood:

$$P(s_i)^t = 1 - (\lambda'_i)^{\frac{\lambda_i}{1-\max(\Delta_i, \Delta'_i)}} \quad (4)$$

Note that the fault history is assessed according to $\max(\Delta_A, \Delta'_A)$. Then, the likelihood of a service executing without any fault is $1 - \max(\Delta_A, \Delta'_A)$. We use this value in relation to the total execution times (remaining given by λ' , and overall given by λ) to assess the likelihood of a service executing *without* a fault. To get the likelihood of the service's fault occurrence we subtract this value from 1 in Equation 4. In cases where we need to incorporate a service's priority weight, Equation 4 becomes:

$$P(s_i)^t = 1 - (\lambda'_i)^{\frac{\lambda_i f(s_i)}{1-\max(\Delta_i, \Delta'_i)}} \quad (5)$$

We observe that with increased service priority, fault likelihood also increases. Based on the fault likelihood, FOLT decides when to build a recovery plan. Services with a high priority are usually critical, and a fault in any of those services may harm the overall QoS. Thus, fault likelihood and service priority are directly proportional in FOLT.

Using Equation 4 as the basis, we define fault likelihood estimation for each invocation model. For instance, the likelihood of fault(s) in a sequential invocation (P_{seq}) is dependent on the successor service(s) [3]. Since FOLT uses invocation points, only a single service can be invoked per time instance/invocation point. Hence the equation stays the same. Let A be the successor service, then

$$P_{seq} = P(s_A)^t = 1 - (\lambda'_A)^{\frac{\lambda_A f(s_A)}{1-\max(\Delta_A, \Delta'_A)}} \quad (6)$$

In a parallel invocation, fault estimation at the invocation point translates to the fault occurring in either of the invoked services. Since all services are independent, we need to add their fault likelihoods. Moreover, due to the likelihood of simultaneous faults occurring in the respective services, we have

$$P_{par} = \bigcup_{i=1}^h P_i = \sum_{i=1}^h P_i - \prod_{i=1}^h P_i \quad (7)$$

$$P_{par} = \sum_{i=1}^h (1 - (\lambda'_i)^{\frac{\lambda_i f(s_i)}{1-\max(\Delta_i, \Delta'_i)}}) - \prod_{i=1}^h (1 - (\lambda'_i)^{\frac{\lambda_i f(s_i)}{1-\max(\Delta_i, \Delta'_i)}}) \quad (8)$$

where h is the number of services invoked in parallel.

In probabilistic invocation (P_{pro}), fault likelihood depends on the probability of selecting the service (Q). Then, if we have k services:

$$P_{pro} = \bigcap_{i=1}^k P_i = \prod_{i=1}^k Q_i \times P_i \quad (9)$$

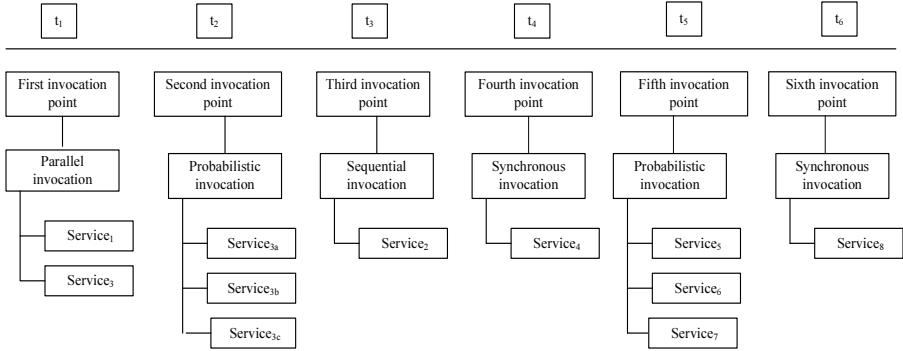


Fig. 5. Simulation Environment of Eleven Services and Six Invocation Points

Similarly, the fault likelihood of a circular invocation is:

$$P_{cir} = \prod_{i=1}^n P_S \quad (10)$$

4 Assessment

We developed a simulator and conducted experiments to analyze the performance of our proposed framework. Our development environment consists of a Windows server 2008 (SP2) based Quad core machine with 8.0 GB of ram. We developed our scenarios using Asp.Net running on Microsoft .Net version 3.5 and SQL as the back-end database. We simulated a services-based system complete with fault prediction, recovery strategies and performance measurement. The input to the system is an XML schema of the system that is used to exhibit the characteristics of a running system.

The experimental results based on the scenario of Figure 2 are discussed below. We are focusing in our experiment in reducing the total execution time, since the service will not be executed if there is a high likelihood that it fails at run time, as this increases the total execution time. In Figure 5, we show a system with 11 services and 6 invocation points. The invocation points are at $t_1 = 30$ ms, $t_2 = 50$ ms, $t_3 = 450$ ms, $t_4 = 560$ ms, $t_5 = 670$ ms, $t_6 = 890$ ms with the total execution time (T) as 1000 ms. At t_1 the system invokes two services ($service_1, service_3$) in parallel, at t_2 the system uses probabilistic invocation for three services ($service_{3a}, service_{3b}, service_{3c}$). At t_3 the system invokes one service ($service_2$) which is sequential invocation, and at t_4 the invocation is synchronous for one service $service_4$. At t_5 the invocation is again probabilistic for three services ($service_5, service_6, service_7$) and at t_6 the system invokes one service ($service_8$). Table 2 shows a sample (i.e. these are not constant) of the different parameter values for all 6 invocation points.

We assume the different theta values for this experiment i.e., $\theta_1 = 0.50$, $\theta_2 = 0.60$. The table lists the priority of each service involved, the services' time

Table 2. Service Parameters at Invocation Points

Invocation Point	Service	Time	Priority	λ_i	λ'_i	Δ_i	Δ'_i	$P(s_i)$
1	<i>Service</i> ₁	180	60%	0.18	0.1856	0.30	0.15	0.2289
1	<i>Service</i> ₃	250	40%	0.25	0.2577	0.40	0.60	0.2875
2	<i>Service</i> _{3a}	80	70%	0.08	0.8421	0.90	0.70	0.7498
2	<i>Service</i> _{3b}	90	50%	0.09	0.0947	0	0.20	0.1241
2	<i>Service</i> _{3c}	80	30%	0.08	0.0842	0.50	0.40	0.1120
3	<i>Service</i> ₂	150	80%	0.15	0.2727	0.70	0.80	0.5414
4	<i>Service</i> ₄	1000	80%	0.1	0.2273	0.60	0.80	0.4471
5	<i>Service</i> ₅	80	90%	0.08	0.2424	0.70	0.65	0.2883
5	<i>Service</i> ₆	70	80%	0.07	0.2121	0.50	0.80	0.3522
5	<i>Service</i> ₇	80	90%	0.08	0.2424	0.75	0.90	0.6395
6	<i>Service</i> ₈	100	80%	0.1	0.9091	0.70	0.80	0.0374

weights and their history ratios (from internal and external experiences). Using Equation 8, FOLT calculated the fault likelihood at the first invocation point (Parallel invocation) to be $P_{par} = 0.4505$. Since *service*₁ and *service*₃ had a very low fault likelihood, this in turn implied that the invocation point fault likelihood was lower. In this case $P_{par} < \theta_1$, FOLT did not build any plan and continued with the system execution. For the second invocation point (Probabilistic invocation), as per the given parameters FOLT calculated the fault likelihood using Equation 9 to be $P_{pro}=0.0104$. Hence, the system did not build a recovery plan and continued its execution. Similarly at third invocation point (Sequential invocation): the fault likelihood was calculated using Equation 6 to be $P_{seq}=0.5414$. In this case $P_{seq} > \theta_1$, the system did build a recovery plan and continued its execution. However, the execution of the created plan had to wait until the occurrence of fault because $P_{seq} < \theta_2$. Fourth invocation point (Synchronous invocation): The fault likelihood was same as of *service*₄ = 0.4471. Fifth invocation point (Probabilistic invocation): The fault likelihood calculated by FOLT was $P_{pro}=0.0649$. Since *service*₇ had a high fault likelihood and the other two services had low fault likelihood , this in turn implied that the invocation point fault likelihood was lower. In the case that the selected service was *service*₇, the system would have build a recovery plan and executed it(fault likelihood of *service*₇ > θ_2). Sixth invocation point (Asynchronous invocation): The fault likelihood of this invocation point was same as that of *service*₈ = 0.0374. For this invocation point the system did not create any plan.

In Figure 6-(a) We can see the eleven services in this system and their fault likelihoods. We notice that *service*_{3a} has the highest fault likelihood and *service*₈ has the lowest fault likelihoods. These results are based on the different service's weight, history, behavior, invocation time and priority. Figure 6-(b) shows the fault likelihood for each invocation point where the highest fault likelihood was at t_3 and the lowest fault likelihood was at t_2 . Figure 6-(c) shows the relationship between the priority and the fault likelihood. For example, *service*_{3a} has a priority of 70% and the fault likelihood is 0.7498, however, the priority for *service*₇

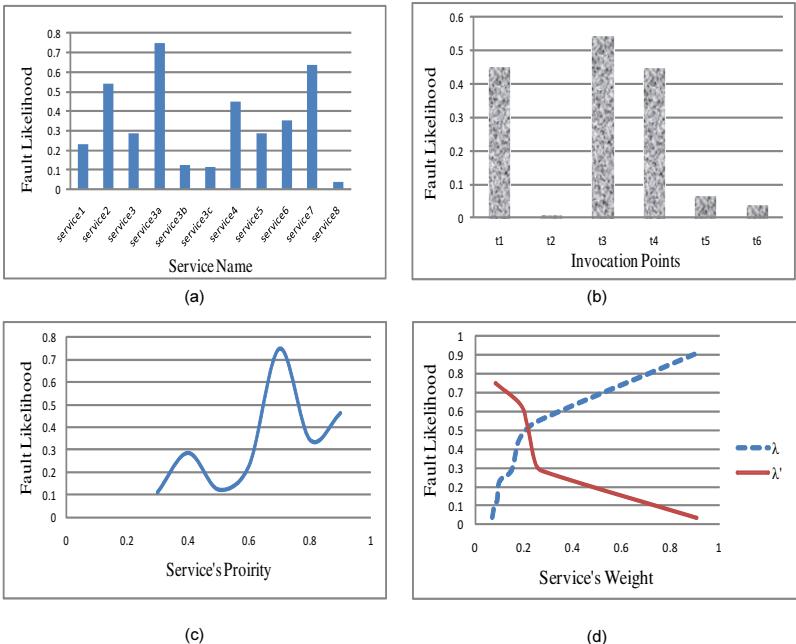


Fig. 6. (a) Services Fault Likelihood (b) Invocations Fault Likelihood (c) Service Priority (d) Service Time Weight

is 90% and the fault likelihood is 0.6395, because it has lower weight. Figure 6-(d) presents the relationship between service weight and the fault occurrence likelihood.

We also performed experiments to assess the FOLT approach's efficiency. Figure 7-(a) shows the comparison between FOLT, no fault and systems that use replace, retry and restart as recovery techniques. Here total execution time is plotted on the y-axis and the number of faults on the x-axis. With increasing number of faults, the execution time also increases. However, FOLT takes less time than compared techniques. This is due to the fact that FOLT preempts a fault and builds a recovery plan for it. Figure 7-(b) shows the total execution time comparisons for the five systems. Here we fix the number of faults to four.

5 Related Work

In this section, we provide a brief overview of related literature on fault management and fault tolerance techniques in service-oriented environments, and the Web in general. Santos et al. [14] proposed a fault tolerance approach (FTWeb) that relies on active replicas. FTWeb uses a sequencer approach to group the different replicas in order. It aims at finding fault free replica(s) for delegating the receiving, execution and request replies to them. FTWeb is based on the

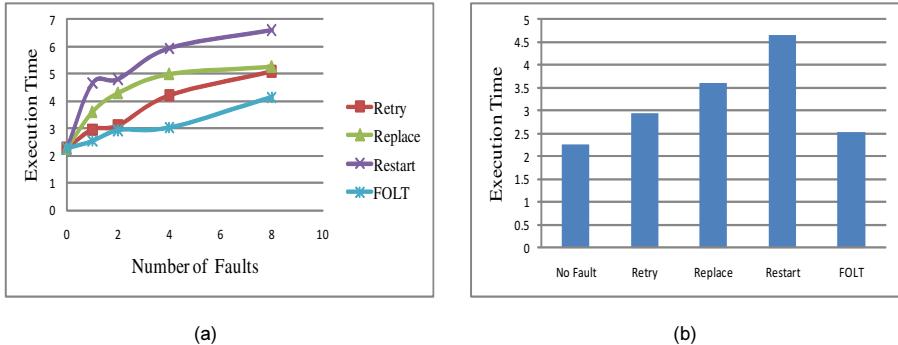


Fig. 7. (a) Total Execution Time Comparisons in Relation to Number of Faults (b) Execution Time Comparisons

WSDispatcher engine, which contains components responsible for: creating fault free service groups, detecting faults, recovering from faults, establish a voting mechanism for replica selection, and invoking the service replicas. Raz et al. [13] present a semantic anomaly detection technique for SOAs. When a fault occurs, it is corrected by comparing the application state to three copies of the service code and data that is injected at a host upon its arrival. Similarly, Hwang et al. [5] analyze the different QoS attributes of web services through a probability based model. The challenge in this approach is composing an alternate work flow in a large search space (with the least error). Online monitoring (for QoS attributes) also needs some investigation in this approach.

Wang et al's. [16] approach integrates handling of business constraint violations with runtime environment faults for dynamic service composition. The approach is divided into three phases. The first phase is defining the fault taxonomy by dividing the faults into four groups (functional context fault, QoS context fault, domain context fault and platform context fault) and analyzing the fault to determine a remedial strategy. The second phase is defining remedial strategies (remedies are selected and applied dynamically). The remedial strategies are categorized into goal-preserving strategies to recover from faults (ignore, retry, replace and recompose) and non-goal preserving strategies to support the system with actions to assist possible future faults (log, alert and suspend). The third phase is matching each fault category with remedial strategies based on the data levels. The main challenge in this approach is the extra overhead, especially when the selected strategy is a “recomposition” of the whole system.

Simmonds et al. [15] present a framework that guarantees safety and liveness through the conversation between patterns, and checking their behaviors. The framework is divided in two parts: (1) Websphere runtime monitoring with property manager and monitoring manager. The property manager consists of graphical tools to transfer the sequential diagram to NFAs and check the XML file. The monitoring manager builds the automata and processes the events. (2) Websphere runtime engine. It uses the built-in service component that already exists in BPEL,

to provide service information at runtime. Delivering reliable service compositions over unreliable services is a challenging problem. Liu et al. [6] proposed a hybrid fault-tolerant mechanism (FACTS) that combines exception handling and transaction techniques to improve the reliability of composite services.

6 Conclusion

We presented a new framework for fault management in service-oriented architectures. Our proposed approach Fault Occurrence Likelihood esTimation (FOLT) depends on the past behavior of services, the invocation method of the services, the execution times of services, and the priority of a specific service in the current system. We identified new metrics to measure the fault occurrence likelihood. We evaluated FOLT using simulations and the results indicate the approach's efficiency and ability to recover from faults. FOLT reduces the overall system execution time by replacing the traditional system recovery methods (i.e., restarting the system at check points) by reacting to the faults by expecting the faults ahead of time and pro-actively building the prevention/recovery plans. In the future, we plan to compare FOLT with other similar existing approaches. In this paper, we presented Phase 1 of our approach, and are currently working on Phase 2 (i.e. generating a fault recovery plan) and Phase 3 (i.e. assessing overall system reliability to see when to execute a plan).

References

1. Abramowicz, W., Haniewicz, K., Kaczmarek, M., Zyskowski, D.: Architecture for web services filtering and clustering. In: International Conference on Internet and Web Applications and Services, p.18 (2007)
2. Bai, C.G., Hu, Q.P., Xie, M., Ng, S.H.: Software failure prediction based on a markov bayesian network model. *J. Syst. Softw.* 74(3), 275–282 (2005)
3. Cardoso, J., Miller, J., Sheth, A., Arnold, J.: Modeling quality of service for workflows and web service processes. *Journal of Web Semantics* 1, 281–308 (2002)
4. D'Mello, D.A., Ananthanarayana, V.S.: A tree structure for web service compositions. In: COMPUTE 2009: Proceedings of the 2nd Bangalore Annual Computer Conference, pp. 1–4. ACM, New York (2009)
5. Hwang, S.-Y., Wang, H., Tang, J., Srivastava, J.: A probabilistic approach to modeling and estimating the qos of web-services-based workflows. *Inf. Sci.* 177(23), 5484–5503 (2007)
6. Liu, A., Li, Q., Huang, L., Xiao, M.: Facts: A framework for fault-tolerant composition of transactional web services. *IEEE Transactions on Services Computing* 99(PrePrints), 46–59 (2009)
7. Malik, Z., Akbar, I., Bouguettaya, A.: Web services reputation assessment using a hidden markov model. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 576–591. Springer, Heidelberg (2009)
8. Malik, Z., Bouguettaya, A.: Ratewebs: Reputation assessment for trust establishment among web services. *The VLDB Journal* 18(4), 885–911 (2009)
9. Menasce, D.A.: Composing web services: A qos view. *IEEE Internet Computing* 8, 88–90 (2004)

10. Meulenhoff, P.J., Ostendorf, D.R., Živković, M., Meeuwissen, H.B., Gijsen, B.M.M.: Intelligent overload control for composite web services. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 34–49. Springer, Heidelberg (2009)
11. Papazoglou, M.: Web Services: Principles and Technology. Pearson-Prentice Hall, London (2008) ISBN: 978-0-321-15555-9
12. Chen, H.p., Zhang, C.: A fault detection mechanism for service-oriented architecture based on queueing theory. International Conference on Computer and Information Technology, 1071–1076 (2007)
13. Raz, O., Koopman, P., Shaw, M.: Semantic anomaly detection in online data sources. In: ICSE 2002: Proceedings of the 24th International Conference on Software Engineering, pp. 302–312. ACM, New York (2002)
14. Santos, G.T., Lung, L.C., Montez, C.: Ftweb: A fault tolerant infrastructure for web services. In: Proceedings of the IEEE International Enterprise Computing Conference, pp. 95–105. IEEE Computer Society, Los Alamitos (2005)
15. Simmonds, J., Gan, Y., Chechik, M., Nejati, S., O'Farrell, B., Litani, E., Waterhouse, J.: Runtime monitoring of web service conversations. IEEE Transactions on Services Computing 99(PrePrints), 223–244 (2009)
16. Wang, M., Bandara, K.Y., Pahl, C.: Integrated constraint violation handling for dynamic service composition. In: SCC 2009: Proceedings of the 2009 IEEE International Conference on Services Computing, pp. 168–175. IEEE Computer Society, Washington, DC, USA (2009)

A Strategy for Efficient Crawling of Rich Internet Applications

Kamara Benjamin¹, Gregor von Bochmann¹, Mustafa Emre Dincturk¹,
Guy-Vincent Jourdan¹, and Iosif Viorel Onut²

¹ SITE, University of Ottawa. 800 King Edward Avenue,
K1N 6N5, Ottawa, ON, Canada

² Research and Development, IBM® Rational® AppScan® Enterprise, IBM, 1 Hines Rd, Ottawa,
ON, Canada

{bochmann, gvj}@site.uottawa.ca,
{kbenj067, mdinc075}@uottawa.ca,
vioonut@ca.ibm.com

Abstract. New web application development technologies such as Ajax, Flex or Silverlight result in so-called Rich Internet Applications (RIAs) that provide enhanced responsiveness, but introduce new challenges for crawling that cannot be addressed by the traditional crawlers. This paper describes a novel crawling technique for RIAs. The technique first generates an optimal crawling strategy for an anticipated model of the crawled RIA by aiming at discovering new states as quickly as possible. As the strategy is executed, if the discovered portion of the actual model of the application deviates from the anticipated model, the anticipated model and the strategy are updated to conform to the actual model. We compare the performance of our technique to a number of existing ones as well as depth-first and breadth-first crawling on some Ajax test applications. The results show that our technique has a better performance often with a faster rate of state discovery.

Keywords: Rich Internet Applications, Crawling, Web Application Modeling.

1 Introduction

Web Applications have been used for nearly as long as the Web itself. In the formative years, a Web Application was an application running entirely on the server side, and which used a dynamically created web page rendered inside a classical web browser as a client. Following the model of static Web sites, these web applications used the usual GET and POST HTTP requests to communicate between the client and the server, and each response sent by the server was meant as a complete replacement of the currently displayed client side. These applications were not very user-friendly due to the lack of flexibility at the client side, but they were pretty similar to static web sites in terms of crawling (except for possible user inputs): it was sufficient to scan the current page to find out what the possible next states could be, and the current state was entirely defined by the last server response. Crawling these web

applications was easy, and many commercial tools can do it, e.g. for the purpose of automated testing, usability assessment, security evaluation or simply content indexing.

This situation was first modified with the widespread support in Web browsers for scripting languages such as JavaScript, and the ability given to these scripts by browsers to modify directly the current page (the current DOM). After this, it was no longer true that the state of the client could be inferred simply from the last server response, since script embedded in this response could have modified the state, perhaps based on user input. This new situation seriously impaired the ability of existing tools to truly crawl these applications. But it was the introduction of another possibility for these scripts that was a real challenge for crawlers: the ability for scripts to asynchronously exchange messages with the server (still using GET and POST commands) without having to reload the page entirely. This technique, most commonly used with Ajax, means that these Web Applications can really be seen as two programs running concurrently in the server and the browser, communicating by asynchronous message exchanges. We call these types of Web Applications “Rich Internet Applications” (RIAs). To the best of our knowledge, there are no tools today to efficiently and systematically crawl RIAs. Some academic tools have been developed [11,17] but the results are not entirely satisfactory yet. Some large commercial software do require crawling abilities, such as vulnerabilities scanners (e.g. see [3] for a recent survey of security scanners), but again their ability to really crawl RIAs is very limited in practice.

It is important to note that the difficulties introduced by RIAs for crawling have a much deeper impact than preventing security testing tools to perform well. First, crawling is the basis for indexing, which is necessary for searching. Our inability to crawl RIAs means that these RIAs are not properly indexed by search engines, arguably one of the most important features of the Web.¹ Second, because RIAs are often more user-friendly than simple sites and provide an enhanced experience for end-users, a growing number of web authoring tools automatically add RIA-type code (usually Ajax) into the produced site. So the problem extends much beyond advanced Web Applications. Even the most classical Web site, built without any programming ability by some content editor, might end up being non-crawlable and thus non-searchable. This is clearly an important problem that must be addressed. This paper is a first step in this direction. For simplicity, we only consider Ajax based RIAs in the following, but the same concepts can be applied to other technologies.

The paper is organized as follows: in Section 2, we provide a general overview of our crawling strategy. In Section 3, we give a high-level description of the set of algorithms we have developed for this purpose. In Section 4, we show how all of these algorithms are tied together to lead to our crawling strategy. In Section 5, we provide experimental results obtained with our prototype, compared with existing tools and other, simpler crawling strategies. Related works is reviewed in Section 6, and we conclude in Section 7 with some future research directions.

¹ To observe the current RIA crawling capabilities of the search engines, we created various experimental RIAs, each with 64 states reachable via AJAX calls. To date, Google and Yahoo visited our applications: neither was able to explore beyond the initial state!

2 Overview of Strategy

The purpose of crawling a RIA is to infer automatically an accurate model of the application. We conceptualize the model of the application as a Finite State Machine. The states of the state machine are the distinct DOMs that can be reached on the client side and a transition is an event execution (e.g. any JavaScript event such as onClick, onMouseOver, onSubmit etc.) on the current DOM. To produce an accurate and useful model, the crawling strategy must satisfy certain requirements. First, the produced model must be complete. That is, the model must represent all reachable states and all transitions. In addition to the event being executed, the next state of a transition may depend on other data, such as user inputs and variable values. Therefore, building a complete model requires capturing all the states and all such events along with the relevant data. Second, the model must be built in a deterministic way. That is, crawling the application twice under the same conditions (including server-side data) should always produce the same model. Third, the model should be produced efficiently. By this, we mean that the model construction strategy should produce as much of the model as possible within the shortest amount of time. Crawling an application may take a very long time, and even can be infinite theoretically; thus it may not be practical or even feasible to wait until the crawling ends. In such a case, if the crawl is stopped before it is complete, the partial model produced should contain as much information as possible.

Satisfying the completeness requirement is not too difficult on its own. There are several simple strategies that will achieve this. One must simply be careful not to overlook any of the intermediate states [4], even though most state-of-the-art crawling strategies fail to meet this requirement, usually on purpose since it is considered to take too much time. The determinism requirement is also not very difficult to achieve, although many RIAs may not easily provide the necessary uniform responses to the same requests. The efficiency requirement is obviously the most challenging one. Even the most simple-minded strategies, such as enumerating and executing one by one all possible orderings of events enabled at the state being explored, may satisfy the first two requirements, but not the efficiency requirement. Considering a single state with n enabled events, there are $n!$ possible different orders for executing these events, so blindly executing events in every possible order is not efficient. However, note that this simple strategy makes no hypothesis about the application beforehand, so there is actually not much room for generating a more efficient strategy as such.

To generate an efficient strategy, we use the following methodology which we call “model-based crawling”:

- First, general hypotheses regarding the behavior of the application are elaborated. The idea is to assume that, in general, the application will usually behave in a certain way, described by the hypotheses. With these hypotheses, one can anticipate what the final model (or part of it) will be, if they are indeed valid. The crawling activity is thus morphed from a discovery activity (“what is the model?”) into a confirmation activity (“is the anticipated model correct?”). Note that the hypotheses do not need to be actually valid.
- Once reasonable hypotheses have been specified, an efficient crawling strategy can be created based on these hypotheses. Without any assumptions about the behavior of the application, it is impossible to have a strategy that is expected to be efficient.

But if one anticipates the behavior, then it becomes possible to draw an efficient strategy to confirm the hypothesis. In our case, we provide a crawling strategy that is in fact optimal for our hypotheses.

- Finally, a strategy must also be established to reconcile the predicted model with the reality of the application that is crawled. Whenever the application's behavior doesn't conform to the hypotheses, some corrective actions must be taken to adapt the model and to update the crawling strategy. In our case, we always assume that the part of the application that has not yet been crawled will obey the initial set of hypotheses, and the crawling strategy is adapted accordingly.

The event-based crawling strategy exposed in this paper follows this methodology. To form an anticipated model, we make following two hypotheses only:

- **H1:** The events that are enabled at a state are pair-wise independent. That is at a state with a set $\{e_1, e_2, \dots, e_n\}$ of n enabled events, executing a given subset of these events leads to the same state regardless of the order of execution.
- **H2:** When an event e_i is executed at state s , the set of events that are enabled at the reached state is the same as the events enabled at s except for e_i .

We believe that these hypotheses are usually reasonable for most of the web applications in the sense that executing different subsets of events is more likely to lead to new states, whereas executing a given subset of events in different orders is more likely to lead to the same state. More precisely, it is obviously not always the case that concurrently enabled events are independent of each other, and that executing an event will not enable new ones or disable existing ones. Still, it is reasonable to assume that, in general, the events that are concurrently enabled are "roughly" independents, as a whole. In fact, if a client state tends to have a large number of events, then it seems reasonable to expect that many of these events are independent of each other, and so even if the hypotheses are often violated for chosen pairs of events, they are usually verified overall. And of course, when this is not the case, the expected model and the corresponding strategy is updated without throwing away the whole strategy.

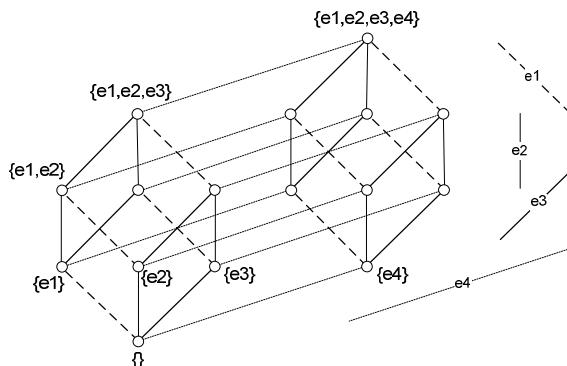


Fig. 1. Hypercube of dimension 4

With these hypotheses, the expected model for a state with n enabled events is a hypercube of dimension n . There are 2^n possible subsets of n events and a hypercube is a partial order of all possible subsets ordered by inclusion. Figure 1 shows an example hypercube of dimension $n = 4$. In a hypercube of dimension n , each of the 2^n states is characterized by a subset of n events executed to reach it. Each edge corresponds to a transition that is triggered by execution of one of the events. There are $n!$ different paths from the bottom state to the top state. Each one of them represents an order of execution containing n events.

Based on the hypercube hypothesis, a strategy should be generated. To satisfy our efficiency requirement, the strategy must aim at visiting each state in the expected model as soon as possible. Once each state has been visited, then the strategy completes the crawl by aiming at efficiently traversing every transition that has not yet been traversed. In the following section, we introduce an algorithm that generates an optimal strategy which completes crawling in $(n \text{ choose } (n/2))^{*}[n/2]$ paths instead of going over $n!$ paths. What is more only $(n \text{ choose } (n/2))$ of these paths are enough to visit all the states in the hypercube. Both numbers are optimal.

Another important factor for the efficiency of the crawling algorithm is the choice of the state equivalence relation. Equivalence relation is used to determine whether a state should be regarded as being the same as another, already known. This avoids exploring the same state multiple times. Our crawling algorithm assumes that some appropriate equivalence relation is provided. Different equivalence relations can be employed depending on the purpose of model. For example, for security scanning, an equivalence relation that ignores text content of pages may be appropriate whereas for indexing the text must be taken into account. An in-depth discussion of this topic is beyond the scope of this paper, but it is important to note that the equivalence function we are working with is assumed to be a real equivalence relation (from the mathematical viewpoint) and that this function should at least be coherent with state transitions, that is, two equivalent states must at least have the same set of embedded URLs (an exception can be made if some specific parts of the page are ignored during the crawl, e.g. a part containing ads for example. URLs and events embedded into the ignored parts can also be ignored) and the same set of events. One obvious such equivalence relation is simple equality. Throughout this paper \approx is used to denote the equivalence relation that is provided to the crawling algorithm.

3 Minimal Transition Coverage (MTC) of a Hypercube

In this section, we present an algorithm to find a minimal set of paths that traverse each transition in a given hypercube. We call such a set a Minimal Transition Coverage (MTC) of the hypercube. Since we also want to be able to visit each state of the hypercube as early as possible, we need more than just any MTC; we need one that will also reach all the states as fast as possible. For this reason, our MTC algorithm can be constrained by a set of disjoint chains of transitions given as input. Each chain of the constraint set will be included in one of the produced MTC chains. When the produced MTC is constrained with a minimal set of chains that visit each state in the hypercube, the resulting MTC is exactly what we are aiming for: a minimum number of paths containing every possible transition of the hypercube and

including as a subset a minimum number of paths visiting every state. We then simply have to give priority to the paths visiting the states to obtain our strategy.

In [5], it was stated that a Minimal Chain Decomposition (MCD) of a hypercube, which is the minimal set of paths visiting every state of the hypercube, could be used as part of a crawling strategy. Here, we use an MCD as a constraint for MTC to generate our strategy. First, we give a short review of the MCD for completeness.

3.1 Minimal Chain Decomposition

Since a hypercube represents a partially ordered set, a path in the hypercube consists of pair-wise comparable elements. A path is also called as a chain of the order. Finding a set of chains covering every element of the order is known as chain decomposition of the order. A minimal chain decomposition of the order is a chain decomposition containing the minimum number of chains (see [1] for an overview of the concepts). In 1950, Dilworth proved that the cardinality of any minimal chain decomposition is equal to the maximum number of pairwise incomparable elements of the order, also known as the width of the order [9]. For a hypercube of size n , the width is $(n \text{ choose } n/2)$. An algorithm that can be used for decomposing a hypercube is given in [8]. Below we present the algorithm as it is exposed in [12] (adapted to our hypercube definition).

Definition (adapted from [12]): The Canonical Symmetric Chain Decomposition (CSCD) of a hypercube of dimension n is given by the following recursive definition:

1. The CSCD of a hypercube of size 0 contains the single chain (\emptyset).
2. For $n \geq 1$, the CSCD of a hypercube of dimension n contains precisely the following chains:

- For every chain $A_0 < \dots < A_k$ in the CSCD of a hypercube of dimension $n - 1$ with $k > 0$, the CSCD of a hypercube of dimension n contains the chains:

$$\begin{aligned} A_0 &< \dots < A_k < A_k \cup \{n\} \text{ and} \\ A_0 \cup \{n\} &< \dots < A_{k-1} \cup \{n\}. \end{aligned}$$

- For every chain A_0 of length 1 in the CSCD of a hypercube of dimension $n - 1$, the CSCD of a hypercube of dimension n contains the chain:

$$A_0 < A_0 \cup \{n\}$$

3.2 MTC Algorithm

In the rest of this paper, we represent a chain C as alternation of states and events in the form $C = s_1 - e_1 - s_2 - e_2 - s_3 - \dots - e_m - s_{m+1}$. Each chain starts and ends with a state. The first state in C is noted by $\text{first}(C)$ and the last state in C is noted by $\text{last}(C)$. Each event e_i in C corresponds to a transition $(s_i - e_i - s_{i+1})$ from s_i to s_{i+1} . The length of the chain C is defined to be the number of states in C and denoted as $|C|$. $\text{pref}_C(s_i)$ denotes the chain that is the prefix of C such that $\text{last}(\text{pref}_C(s_i))$ is s_i . $\text{suffix}_C(s_i)$ denotes the chain that is the suffix of C such that $\text{first}(\text{suffix}_C(s_i))$ is s_i . Two chains C_1 and C_2 can be concatenated if $\text{last}(C_1) = \text{first}(C_2)$. The resulting chain is represented as $C_1 + C_2$.

For a given hypercube of dimension n , we present an algorithm to generate an MTC in three stages. The algorithm allows MTC to be optionally constrained by a set

(C_C) of disjoint chains of the transitions. We call level 0 the bottom of hypercube, level n the top of the hypercube and level $[n/2]$ middle of the hypercube.

Upper Chains Stage: In this stage, a set of chains (C_U) covering all the transitions above the middle of the hypercube is calculated. For each transition leaving a state at the middle level, a chain is constructed by extending the transition toward the top of the hypercube. Let U be the chain that is currently being constructed and s be last(U). Initially, U is a chain containing only the middle level state. Then we pick an unvisited transition ($t = s - e - s'$) leaving s and attempt to extend U with t . If t is not part of any constraint chain then we can extend U with t . Otherwise t is included in a constraint chain $C \in C_C$. There can be two cases,

- If $|U| = 1$ or s is first(C) \rightarrow extend U with suff $_C(s)$.
- Otherwise we cannot use t to extend U . So, we mark t as not usable and we attempt to extend U by picking another unvisited transition.

The extension of U continues until no transition can be found to extend it further.

Lower Chains Stage: In this stage, a set of chains (C_D) covering all the transitions below the middle level of the hypercube is calculated. This stage is simply the symmetric of the upper chains stage.

Chain Combination Stage: In the final stage, the chains in C_U and C_D are combined into larger chains. Note that, when the hypercube has an odd dimension, the number of transitions entering a middle-level state is one less than the number of transitions leaving it. In that case, for some upper chains, we cannot find a lower chain to combine with. For this reason, in this stage we iterate over the lower chains and try to combine each with an upper chain. After all lower chains combined, any uncombined upper chains are simply added to the resulting set. Again the only consideration while combining a lower chain with an upper chain is to keep every constraint chain as it is. That is, if C is a constraint chain $C \in C_C$ has a prefix in the lower half and a suffix in the upper half, then the lower chain D containing the prefix of C can only be combined with the upper chain U containing the suffix of C and vice versa.

Note that, the number of chains in an MTC of a hypercube of size n equals $(n \text{ choose } (n/2)) * [n/2]$ which is the number of states in the middle level (or the width of the hypercube) multiplied by the number of transitions leaving each middle level state.

4 Overall Strategy for Crawling RIAs

There are two methods for reaching new client states while crawling a RIA. One is through making synchronous HTTP requests to the server via URLs embedded in the DOM. The other is executing events in the DOM. An overall strategy for crawling RIAs should be a mix of these two and possibly running them alternately. For lack of space, we do not elaborate here on the traditional, URL crawling part of the strategy, nor on the way we alternate between URL crawling and event crawling (see [4] for some details), and focus on event-based crawling only.

During URL-based crawling, whenever a page with a non-empty set of events is discovered, it is forwarded to the event based crawler for exploration. If such a state (called a base state) has n events, then the expected model we use to generate the crawling strategy is a hypercube of dimension n based on that state. To explore the states reachable from a base state s , a strategy $\text{Chains}(s)$ that consists of a set of chains is generated. Let $\text{MTC}(s)$ denote a set of MTC chains constrained by an MCD of the hypercube based on s . To generate the “initial” strategy, we use $\text{MTC}(s)$ as a basis. Note that the chains in $\text{MTC}(s)$ may not be executed in their original form because a chain in $\text{MTC}(s)$ does not necessarily start at the base state s . To be able to begin executing events in a chain $C \in \text{MTC}(s)$, we have to reach the state $\text{first}(C)$. For this reason each C must be prepended with a (possibly empty) transfer chain, denoted $T(\text{first}(C))$, from the base state s to $\text{first}(C)$. Any chain can be used as a transfer chain for C as long as it starts from the base state s and reaches $\text{first}(C)$. Thus the initial event execution strategy for the hypercube based on state s will simply be the set $\text{Chains}(s) = \{C' = T(\text{first}(C)) + C \mid C \in \text{MTC}(s)\}$. After, we begin exploration of the base state s , by executing the chains in $\text{Chains}(s)$. Since we aim to reach every state first, we execute the chains that contain MCD chains first.

Executing a chain simply means executing the events of the chain one after the other. After an event is executed, if the state reached is a new state then this state is scanned and the newly found URLs are added to the traditional crawler. Then, we need to check if the state reached has the expected characteristics. If it is not, the current strategy and expected model must be revised.

The expected characteristics of a reached state are determined by the model hypotheses (in our case H1 and H2). H1 has two possible violations (non-mutually exclusive)

- Unexpected Split: In this case, we are expecting to reach a state that has already been visited but the actual state reached is a new state.
- Unexpected Merge: In this case, we unexpectedly reach a known state.

H2 can also be violated in two ways (non- mutually exclusive)

- Appearing Events: There are some enabled events that were not expected to be enabled at the reached state.
- Disappearing Events: Some events that were expected to be enabled at the reached state are not enabled.

When a violation occurs, the revision method creates a brand new strategy (still based on the original hypotheses) for the state causing violations and updates any chain in the existing strategy that is affected by the violations. As a result, the strategy will again be valid for the actual model of the application discovered so far. In the following, we give an outline (see Figure 2) and a high-level overview of the revision method, details can be found in [4].

Consider a hypercube based on state s_1 and the strategy $\text{Chains}(s_1)$ currently used for s_1 . Let the current chain be $C = s_1 - e_1 - s_2 - \dots - s_i - e_i - s_{i+1} - \dots - s_n$ in $\text{Chains}(s_1)$ and let s_i be the current state. According to C we execute the event e_i at s_i . Let s' be the actual state reached after this execution.

First we have to make sure that s' is a state that is in accordance with the expected model. What we expect s' to be depends on whether s_{i+1} is a known state or not. s_{i+1} is a known state if we executed a chain $C' \neq C$ in $\text{Chains}(s_1)$ such that $\text{pref}_C(s_{i+1})$ executes exactly the events $\{e_1, e_2, \dots, e_i\}$ in any order without any violations. If this is the case then we expect s' to be equivalent to s_{i+1} . Otherwise we expect s' to be a new state without any appearing/disappearing events. If the expectations are met, from now on we regard s_{i+1} in the strategy as s' .

If s' is as we expected, the current strategy is still valid but it can be revised by removing some redundant chains. This is possible when C is our first attempt to visit s_{i+1} and it turns out that we had already been in s' . This is either because s' was reached through a base state different than s_1 or s' was reached through s_1 but the first time s' was reached there was an appearing/disappearing event violation. In either case, since s' has already been discovered, there must be an existing strategy to explore it. Hence all the chains having the prefix $\text{pref}_C(s_{i+1})$ are redundant and can be removed from $\text{Chains}(s_1)$.

Procedure reviseStrategy

Let $C = s_1 - e_1 - s_2 - \dots - s_i - e_i - s_{i+1} - \dots - s_n$ be the current chain
 e_i be the event that has just been executed in C
 s' be the state reached by executing event e_i at s_i

If s' meets the expected model characteristics then

- If the current model contains s such that $s' \approx s$ then
- - remove from $\text{Chains}(s_1)$ all chains containing $\text{pref}_C(s_{i+1})$;
- stop procedure reviseStrategy

Else // s' does not meet the expected model characteristics

- For each C' in $\text{Chains}(s_1)$ such that $\text{pref}_{C'}(s_{i+1}) = \text{pref}_C(s_{i+1})$
- - Let s_k be the first state of C' after s_i that belongs to another chain C'' of $\text{Chains}(s_1)$ that does not include $s_i - e_i - s_{i+1}$, if any
- - - add the chain $\text{pref}_{C''}(s_k) + \text{suff}_{C''}(s_k)$ to $\text{Chains}(s_1)$;
- - - remove C'' from $\text{Chains}(s_1)$ if s_k is its last state
- - remove C' from $\text{Chains}(s_1)$
- If this is the first time s' is visited, generate the crawling strategy for s' based on the hypercube hypothesis, with $T(s') = \text{pref}_C(s_i) + (s_i - e_i - s')$

Fig. 2. Revising the Crawling Strategy after executing an event

In case s' is not as expected, revision procedure checks for an alternative way to complete the remaining portion of C as much as possible. In order to do that, we first look for an alternative path among the remaining chains in $\text{Chains}(s_1)$ to reach the minimum indexed state s_k ($i < k < n$) in $\text{suff}_C(s_1)$ without traversing the problematic transition $(s_i - e_i - s_{i+1})$. If a chain C'' has such a prefix then we can use that prefix to reach s_k and execute $\text{suff}_{C''}(s_k)$ after that. That is, we add a new chain $\text{pref}_{C''}(s_k) + \text{suff}_{C''}(s_k)$ to the strategy. Note that if s_k is the last state in C'' then to eliminate redundancy we can remove C'' from the strategy as it is covered completely

in the newly added chain. Also we should remove C from the strategy as it is now handled by newly added chain. This revision should be applied not only to C but to any chain C' having the prefix $\text{pref}_C(s_{i+1})$, since C' will face the same violation.

After all affected chains are updated, we check if we need to generate a strategy for s' . If s' is a state that has been discovered before, then we do not have to do anything as we must have already generated a strategy for s' at the time it was first discovered. Otherwise, a strategy ($\text{Chains}(s')$) should be generated for a new hypercube based on s' . Note that, currently the only known way to reach s' is using chain $\text{pref}_C(s_i) + (s_i - e_i - s')$. For this reason, newly generated hypercube is actually an extension to the existing one and belongs to the same projection based on the state s_1 . Hence each chain in $\text{Chains}(s')$ is added into $\text{Chains}(s_1)$ using $\text{pref}_C(s_i) + (s_i - e_i - s')$ as a transfer chain to reach s' .

5 Experimental Results

To assess the efficiency of our crawling algorithm, we have implemented a prototype tool (see [4] for details). We present results showing the performance of our prototype compared with a prototype of major commercial software for testing web applications, an open source crawling tool for Ajax applications (Crawljax) [17] as well as pure breadth-first and depth-first crawling.

To evaluate a method, we first check if it achieves a complete crawl. For the ones that can crawl all the test applications completely, we record the total number of event executions and the total number of reloads required before discovering all the states and all the transitions separately. The collected data gives us the total number of event executions and reloads required to complete the crawl and an idea about how quickly a method is able to discover new states and transitions.

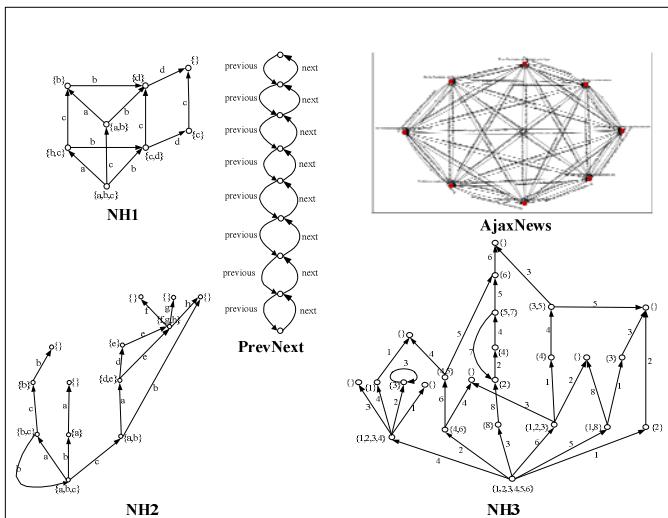


Fig. 3. Models of some test applications

The results were obtained using six Ajax test applications. (See Figure 1 and 3 for the models). Five of them were developed by us and one is a publicly available test application. We designed the applications to reflect different characteristics of the web applications as well as some extreme cases like a perfect hypercube or a complete graph. The first application, called H4D, has the model of a 4 dimensional hypercube (16 states, 32 transitions). Since hypercube is the expected model, H4D is a best case for our strategy. The second application NH1(8 states, 12 transitions) slightly deviates from the hypercube structure. The third and the fourth applications, NH2 (13 states, 15 transitions) and NH3 (24 states, 32 transitions), also deviate from the hypercube structure. The fifth application, PrevNext (9 states, 16 transitions) represents a model that is encountered in many Ajax applications. States of PrevNext are linked by previous and next events. The last application AjaxNews² (8 states, 80 transitions) is a publicly available test site [11]. It is in the form of a news reader with 8 different articles. It allows users to browse articles by previous and next buttons as well as to go directly to an article via clicking titles listed on a menu.

In an effort to minimize any influence that may be caused by considering events in a specific order, the events at each state are randomly ordered for each crawl. Also, each application is crawled 10 times with each method and the average of these 10 crawls are used for comparison.

Only depth-first, breadth-first and our algorithm successfully discovered all the states and all the transitions for all the applications. The commercial product prototype could not achieve a complete crawl for any of the applications. That is because it did not apply a specific strategy for crawling RIAs but blindly executed the events on a page once (in the sequence that they appear) without handling DOM changes which may add or remove events to the DOM. This is an example of a very naïve approach. Crawljax could crawl some completely but not most of them. (This is because Crawljax decides which events to execute at the current state using the difference between the current state and the previous state. That is, if a state transition leaves an HTML element with an attached event unchanged, then that event is not executed at the current state although it is still likely that this event could have led to a different state when executed in the current state). Table 1 shows the number of states discovered by each tool.

Table 1. Number of states discovered by Crawljax, Commercial software and our tool

Application	Total States	States Discovered by		
		Commercial	Crawljax	Our Tool
H4D	16	5	11	16
NH1	8	4	8	8
NH2	13	3	6	13
NH3	24	3	14	24
PrevNext	9	3	3	9

For each test application and for each method, we present the number of transitions and reloads required to visit all the states (and traverse all the transitions) of the application. Figures 5a and 5b shows the total transitions and total reloads required

² <http://www.giannifrey.com/ajax/news.html>

for discovering states, respectively. Figure 6a and 6b shows the total transitions and total reloads required for traversing all transitions. For compactness we use boxplots that should be interpreted as follows. The top of vertical lines show the maximum number required to discover all the states (or all transitions). The lower edge of the box, the line in the box and the higher edge of the box indicate the number required to discover a quarter, half and 3 quarters of all the states (or transitions) in the application, respectively. We will use the position of this box and the middle line to assess whether a method is able to discover new states (or transitions) faster than others.

As Figure 5a shows for H4D, NH1, NH2 and NH3 our method discovered all the states using the least number of transitions. For PrevNext, depth-first required slightly fewer transitions than ours and for AjaxNews breadth-first required significantly fewer transitions. Since the underlying model of AjaxNews is a complete graph, it is an optimal case for breadth-first. Hence it was not surprising to see breadth-first discovering all states using minimal amount of transitions. Looking at the figure to assess the rate of state discovery measured by the number of transitions, we conclude that our method discovers states at a faster rate, except for AjaxNews. Even for AjaxNews, our algorithm discovered more than half of all the states at a rate comparable to breadth-first.

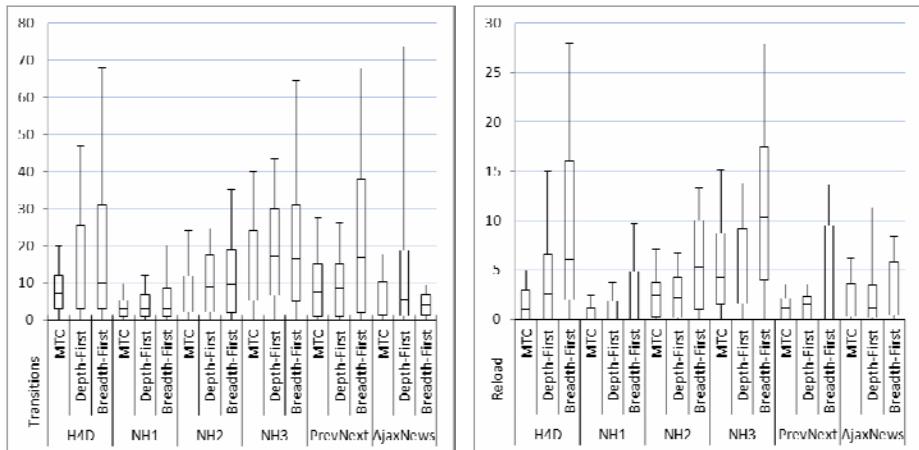


Fig. 5. a. Transitions for state discovery

b. Reloads for state discovery

Figure 5b contains the boxplots of the number of reloads required before discovering all states. It is clear that, our method uses, if not fewer, a comparable number of reloads to depth-first. For H4D and AjaxNews, our method uses significantly less reloads than the others. Looking at the figure to assess the rate of state discovery measured by the number of reloads, conclusion is that our method is able to discover states at higher rates than other mehtods except AjaxNews for which the rate is slightly lower than depth-first.

Figure 6a contains the boxplots showing for each application and for each method the number transitions executed before having traversed all transitions. Comparing the number of transitions used to be able to traverse all the transitions (note that this is the number of transitions executed to complete the crawl), we see that for H4D and NH1 our method is the best, for PrevNext depth-first and our method are even and for NH2 and NH3 depth-first uses less transitions. For AjaxNews breadth-first uses the fewest transitions as expected. Comparing the rate of executing unexecuted transitions, for all the applications except PrevNext, our method uses fewer transitions to execute the majority of transitions. For PrevNext, depth-first has a slightly better rate but the rate is better for ours for more than half of the transitions before depth-first slightly takes over. Note also that, for AjaxNews our method has a significantly higher rate.

Lastly, Figure 6b shows the number of reloads required before executing all transitions. Looking at the total number of reloads required to complete the crawl, for H4D, NH1 and AjaxNews our method was the best, for PrevNext depth-first and our method are even and for NH2 and NH3 depth-first used slightly fewer reloads. Comparing the rates, in order to execute the majority of the transitions, our method required fewer reloads for all the applications except PrevNext, for which depth-first and our algorithm have similar results.

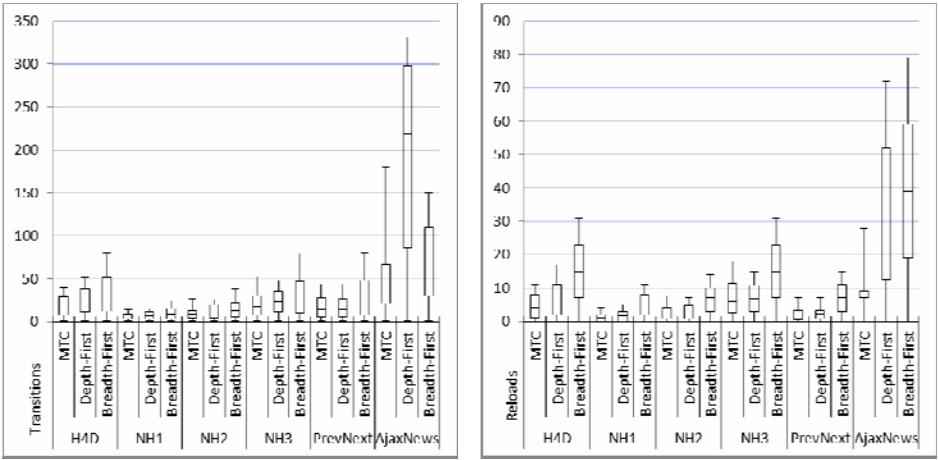


Fig. 6. a. Transitions for new transitions

b. Reloads for new transitions

As an overall evaluation based on these results, we make following remarks. Our approach can extract a correct model of an application regardless of its model. In terms of the total number of transitions and reloads, our approach was the most consistent. That is, for all the cases it was either the best or very close to the best. It was never the worst. This is because, our approach tries to balance the good characteristics of the depth-first and breadth-first approaches. It tries, like depth-first, to maximize the chain length and thus reduce the number of reloads. Also it tries, like breadth-first, to cover the breadth of the application quickly. As we aimed, our strategy has a better rate of discovering new states and transitions.

6 Related Work

We survey here only research focusing on crawling RIAs, for general crawling see [2,7,13]. Although limited, several papers have been published in the area of crawling RIAs, mostly focusing on Ajax applications. For example [10,11,15] focus on crawling for the purpose of indexing and search. In [16], the aim is to make RIAs accessible to search engines that are not Ajax-friendly. In [18] focus is on regression testing of AJAX applications whereas [6] is concerned with security testing.

The works that contain the modeling aspect seem to agree on representing models using finite state machines [14,15,17]. Usually states are defined using the DOM structure and the transitions are based on the events that lead to changes in the DOM [15,17]. In [14], however the states, as well as the events to consider, are abstracted by observing DOM changes in the given execution traces of the application.

As a crawling strategy, [15] uses breadth-first crawl. In order to reduce the number of Ajax requests, whenever a specific Ajax request is made for the first time, the response from the server is cached. Later, if there is a need to call the same function with the same parameters for execution of some event, the cashed response is used instead of making an actual request. This assumes calling the same function with the same parameters always produces the same response regardless of the current state. Therefore it provides no guarantee to produce a correct model. In [17], a depth-first-based method is described and implemented in Crawljax. As we have explained in Section 5, this method, too, is incapable of producing a complete model.

7 Conclusion and Future Work

In this paper, we have presented a general model-based strategy for crawling RIAs. Our solution aims at finding all the client states as soon as possible during the crawl but still eventually finds all the states. Experimental results show that our solution is correct and provides good results on a set of experimental sites: although our solution is not always the best one, it is the only one that works always and seems to always provide one of the best results in terms of efficiency. We believe that these are very encouraging results, but more must be done. We are currently working on the following enhancements:

First, we must extend our experimental results to include large number of real RIAs. We believe that the results will in fact be even more favorable when crawling sites such as Facebook or Google Calendar, because these sites are not going to have the extreme patterns of our test sites, that sometimes favor one crawling technique over the other.

Second, a possible criticism of our solution may be the computation of the strategy based on the hypercube hypothesis ahead of time. Because of the explosion of the size of a hypercube model, this approach may not be practical. Although we have already addressed this issue by generating chains one at a time, as required during the crawl, for simplicity we prefered to explain our solution by using pre-computed chains. We plan to explain the details of the on-the-fly chain generation technique in a separate contribution.

A third enhancement would be to look at other models, beyond the hypercube. Our model-based crawling methodology can be used with any models, and we are currently investigating which other models could be efficiently used.

Finally, a fourth direction would be to enhance the model with the notion of “important” states and events, that is, some states would have priority over others and some events would be more important to execute than others. Again, we believe that our model-based crawling strategy can be adjusted to deal with this situation.

Acknowledgments. This work is supported in part by IBM and the Natural Science and Engineering Research Council of Canada.

Disclaimer. The views expressed in this article are the sole responsibility of the authors and do not necessarily reflect those of IBM.

Trademarks. IBM, Rational and AppScan are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

References

1. Anderson, I.: Combinatorics of Finite Sets. Oxford Univ. Press, London (1987)
2. Arasu, A., Cho, J., Garcia-Molina, A., Paepcke, A., Raghavan, S.: Searching the web. *ACM Transactions on Internet Technology* 1(1), 2–43 (2001)
3. Bau, J., Bursztein, E., Gupta, D., Mitchell, J.C.: State of the Art: Automated Black-Box Web Application Vulnerability Testing. In: Proc. IEEE Symposium on Security and Privacy (2010)
4. Benjamin, K.: A Strategy for Efficient Crawling of Rich Internet Applications. Master’s Thesis. SITE-University of Ottawa (2010), <http://ssrg.site.uottawa.ca/docs/Benjamin-Thesis.pdf>
5. Benjamin, K., Bochmann, G.v., Jourdan, G.V., Onut, I.V.: Some Modeling Challenges when Testing Rich Internet Applications for Security. In: First International Workshop on Modeling and Detection of Vulnerabilities, Paris, France (2010)
6. Bezemer, B., Mesbah, A., Deursen, A.v.: Automated Security Testing of Web Widget Interactions. In: Foundations of Software Engineering Symposium (FSE), pp. 81–90. ACM, New York (2009)
7. Brin, S., Page, L.: The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems* 30(1-7), 107–117 (1998)
8. Bruijn, N.d.G., Tengbergen, C., Kruyswijk, D.: On the set of divisors of a number. *Nieuw Arch. Wisk.* 23, 191–194 (1951)
9. Dilworth, R.P.: A Decomposition Theorem for Partially Ordered Sets. *Annals of Mathematics* 51, 161–166 (1950)
10. Duda, C., Frey, G., Kossmann, D., Zhou, C.: AJAXSearch: Crawling, Indexing and Searching Web 2.0 Applications. VLDB (2008)
11. Frey, G.: Indexing Ajax Web Applications, Master’s Thesis, ETH Zurich (2007)
12. Hsu, T., Logan, M., Shahriari, S., Towse, C.: Partitioning the Boolean Lattice into Chains of Large Minimum Size. *Journal of Combinatorial Theory* 97(1), 62–84 (2002)

13. Manku, G.S., Jain, A., Das Sarma, A.: Detecting near-duplicates for web crawling. In: Proceedings of the 16th International Conference on World Wide Web. WWW 2007, pp. 141–150. ACM, New York (2007)
14. Marchetto, A., Tonella, P., Ricca, F.: State-based testing of Ajax web applications. In: Proc. 1st IEEE Intl. Conf. on Software Testing Verification and Validation (ICST 2008). IEEE Computer Society, Los Alamitos (2008)
15. Matter, R.: Ajax Crawl: making Ajax applications searchable. Master's Thesis. ETH, Zurich (2008)
16. Mesbah, A., Deursen, A.v.: Exposing the Hidden Web Induced by AJAX.TUD-SERG Technical Report Series. TUD-SERG-2008-001 (2008)
17. Mesbah, A., Bozdag, E., Deursen, A.v: Crawling Ajax by Inferring User Interface State Changes. In: Proceedings of the 8th International Conference on Web Engineering, pp. 122–134. IEEE Computer Society, Los Alamitos (2008)
18. Roest, D., Mesbah, A., Deursen, A.v: Regression Testing Ajax Applications: Coping with Dynamism. In: Third International Conference on Software Testing, Verification and Validation, pp. 127–136 (2010)

Graph-Based Search over Web Application Model Repositories

Bojana Bislimovska, Alessandro Bozzon, Marco Brambilla, and Piero Fraternali

Politecnico di Milano, Dipartimento di Elettronica e Informazione
P.zza L. Da Vinci, 32. I-20133 Milano - Italy
`{name.surname}@polimi.it`

Abstract. Model Driven Development may attain substantial productivity gains by exploiting a high level of reuse, across the projects of a same organization or public model repositories. For reuse to take place, developers must be able to perform effective searches across vast collections of models, locate model fragments of potential interest, evaluate the usefulness of the retrieved artifacts and eventually incorporate them in their projects. Given the variety of Web modeling languages, from general purpose to domain specific, from computation independent to platform independent, it is important to implement a search framework capable of harnessing the power of models and of flexibly adapting to the syntax and semantics of the modeling language. In this paper, we explore the use of graph-based similarity search as a tool for expressing queries over model repositories, uniformly represented as collections of labeled graphs. We discuss how the search approach can be parametrized and the impact of the parameters on the perceived quality of the search results.

1 Introduction

Software project repositories are a main asset for modern organizations, as they store the history of competences, solutions, and best practices that have been developed in time. Such invaluable source of knowledge must be easily accessible to analysts and developers through easy and efficient querying mechanisms. This is especially true in the context of Model Driven Development, where models can be reused and shared to improve and simplify the design process, while providing better management and development of software.

Search engine can be a winning solution in this scenario. Source code search engines are already widely adopted and exploit the grammar of the programming languages as an indexing structure of artefacts. Innovative model-based search engines should index their content based on the metamodels of the stored models: Domain Specific Languages (DSL) models, UML models, Business Process Models and others.

Even if traditional keyword querying [8] is the user interaction paradigm for search engines, query by example (content-based) approaches for search have often proven effective in finding results which more closely reflect the user needs

[2]. In the context of model-based search, the query by example interaction can be enacted by providing as queries model or model fragment, so to consider as part of the user information need also the relationships between elements. Such an approach calls for matching and ranking algorithms that rely on techniques such as schema matching [16] or graph matching [7] for finding similarities between models.

The goal of this paper is to propose a general-purpose approach using graph query processing, as a form of querying by example, for searching repository of models represented as graphs. This is realized by performing similarity search using graph matching based on the calculation of graph edit distance. The contribution of this paper includes: 1) a graph based approach for content-based search in model repositories; 2) a flexible indexing strategy adaptive to the relationships among model elements found in the DSL metamodel; 3) a similarity measure that exploits semantic relationships between model element types; and 4) implementation and evaluation of the proposed framework by using projects and queries encoded in a Web Domain Specific Language called WebML (Web Modeling Language)¹. Although the evaluation is performed on a repository of WebML projects, the approach is general and can be applied to any DSL described through a metamodel.

The paper is organized as follows: Section 2 presents the state of the art for repository search engines; Section 3 outlines the fundamentals of model-based search; Section 4 illustrates our graph-based solution for model search; Section 5 discusses the experiments performed on a Domain Specific Language, showing the experiment design and the results obtained; Section 6 draws the conclusions and the future work directions.

2 Related Work

The problem of searching relevant software artifacts in software repositories has been extensively studied in many academic works and widely adopted by the community of developers.

Model search requires some knowledge about the model structure for indexing and querying models. Moogle is a model search engine that uses UML or Domain Specific Language (DSL) metamodels to create indexes for evaluation of complex queries [8]. Our previous work [1] performs model search using textual information retrieval methods and tools, including model segmentation, analysis and indexing; the query language adopted is purely keyword-based. Nowick *et al.* [13] introduce a model search engine that applies a user-centric and dynamic classification scheme to cluster user search terms. Existing approaches performing content-based search rely on graph matching or schema matching techniques. Graphs allow a general representation of model information. Graph matching determines the similarity of a query graph and the repository graphs by matching approximately the query and the model. An indexing approach for business

¹ <http://www.webml.org>

process models based on metric trees (M-Trees), and a similarity metric based on the graph edit distance is illustrated in [7]. *TALE* [19] provides approximate subgraph matching of large graph queries through an indexing method that considers the neighbors of each graph node. Three similarity metrics for querying business process models are presented by *Dijkman et al.* in [4], based on the label matching similarity, structural similarity which also includes the topology of models, and behavioral similarity which besides the element labels considers causal relations in models. The same authors propose the structural similarity approach in [15] which computes similarity of business process models, encoded as graphs, by using four different graph matching algorithms: a greedy algorithm, an exhaustive algorithm with pruning, a process heuristic algorithm, and the A-star algorithm. [2] uses extended subgraph isomorphism for matching approximate queries over databases of graphs.

Schema matching is the correlated problem of finding semantic correspondences between elements of two database schemas [9] [14]. An integrated environment for exploring schema similarity across multiple resolutions using a schema repository is given in [16]. HAMSTER [12] uses clickstream information from a search engine’s query log for unsupervised matching of schema information from a large number of data sources into the schema of a data warehouse.

Other content-based approaches utilize specific algorithms for search. The work in [18] uses domain-independent and domain-specific ontologies for retrieving Web Service from a repository by enriching Web service descriptions with semantic associations. [10] presents a framework for model querying in the business process modeling phase, enabling reuse, support of the decision making, and querying of the model guidelines.

The focus of this paper is on content-based search on web application models, by adopting a graph matching method. For the graph matching we chose the *A-star algorithm* described in [15]. The original work applied *A-star* only to business process models, while our final aim is to propose an approach that is metamodel-aware, and therefore general enough to be applied to different models and languages. We also provide an extensive analysis of the search framework, to show how different parameter settings influence the quality of results.

3 Fundamentals of Model-Based Search

Search of model repositories can be performed in several ways. In *text-based model search*, a project in the repository is represented as an unstructured document composed of (possibly weighted) bag of words; projects are retrieved by matching keyword-based queries to the textual information extracted from a model, i.e., its metamodel grammar and the annotations produced by developers. Facets can be used to allow the user to further filter the information using model properties; advanced result visualization may facilitate the identification of query results within the examined repository [1].

In *content-based model search*, the role of the metamodel is more prominent, as it allows for a finer-grained information retrieval that takes into account the rela-

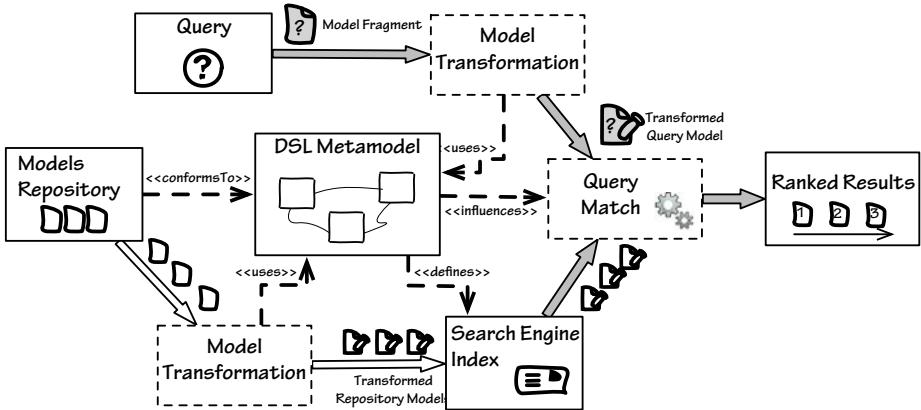


Fig. 1. The content analysis (white arrows) and query (grey arrows) flows of a content-based model search system

tionships between elements, sometimes also considering their semantic similarity. In addition, the metamodel deeply influences the way models are represented in the search engine, as the indexes must contain information about the properties, hierarchies and associations among the model concepts.

Queries over the model repository are expressed using models fragments; a query conforms to the same metamodel as the projects in the repository, and results are ranked by their similarity w.r.t. the query. The DSL metamodel can also influence the search engine matching and ranking operation by providing domain-specific information that can help fine tuning its behavior.

The prominent role of the metamodel demand for a revisit of the classical IR systems architecture: Figure 1 shows the organization of the two main information flows in a *content-based model search* system: the *content processing* and the *query* flows. In a general-purpose content-based search system, the indexing and querying processes are designed to be model-independent, in the sense that they can be configured based on the meta-model of interest. The designer of the system only has to decide which is the information that needs to be extracted, and the meta-model based rules for extraction.

Content processing is applied to the projects in the repository in order to transform them and extract the relevant information for the index creation. Projects are transformed into a suitable representation that captures the model structure and also contains general information about the project and information for each model element, like element name, type, attributes, comments, relationships with other elements, or any other information that can be expressed in a model. Such a transformation is driven by the models' metamodel, and produces as output a metamodel-agnostic representation (e.g., a graph) that is used to build the structured index subsequently used for queries.

The **Query processing** phase of content-based search applies the same transformations to the query as to the projects in the repository, because the query

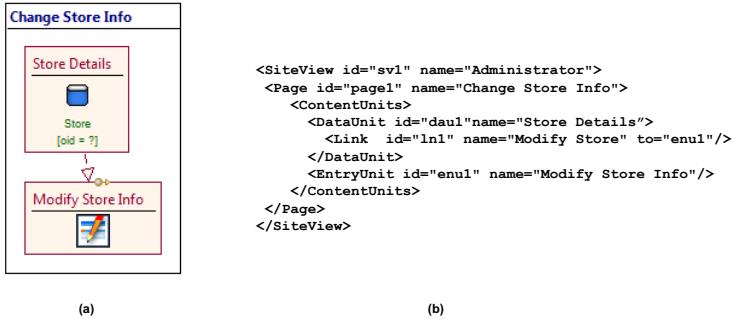


Fig. 2. Example of WebML model (a) and its XML representation (b)

itself is a model or model fragment. Then, the query is matched against the index using different algorithms. As a result, a ranked list of projects is obtained, according to the similarity with the query. Several model comparison techniques can be employed, e.g., based on alternative graph matching approaches.

4 A Graph-Based Approach to Model-Based Search

In this section we elaborate on a general-purpose *content-based model search* approach using query on graph (by example) for searching repository of Web application models represented as graphs. Graphs provide a convenient way for representing data in many different application domains such as computer vision, data mining and information retrieval. To fully exploit the information encoded in graphs, effective and efficient tools are needed for their querying. We implement content-based search by using a graph matching (subgraph isomorphism) technique, to retrieve the most relevant projects from a repository with respect to a query expressed as a model (fragment). We assume that both the models in the repository and the query model conform to the same metamodel.

4.1 Scenario: Repository of WebML Models

To ease the discussion, we exemplify our approach using WebML (Web Modeling Language) [3] models. WebML allows high-level description of a Web site considering different dimensions. The main WebML constructs are pages, units and links, organized into areas and site views. A site view is a coherent hypertext, fulfilling a well-defined set of requirements for a user role. Pages are composed by units. Content units are atomic elements that determine the web page content while operation units support arbitrary business logic triggered by navigation. Units are connected through links forming a hypertext structure.

WebML models can be represented with a graphic notation or with an XML syntax. Figure 2 shows a very simple excerpt of WebML model: the *ChangeStoreInfo* page of the Administrator site view contains a *StoreDetails* data unit that

shows the details about a store, and a link to the *ModifyStoreInfo entry unit* that allows the user to submit new data through a form; a link from the data to the entry unit denotes that the fields of the latter are preloaded with values extracted from the former.

4.2 Graph Representation of Models

Our work exploits graph matching by finding a mapping between two graphs: the query graph and the project graph. Graphs allow abstract representation of models, encoding the specific model features into nodes and edges. Therefore, the models from the repository and the queries are represented as directed annotated graphs, preserving model topologies, and encapsulating the model information as annotations.

In the example scenario, WebML model elements (e.g., siteviews, pages, units) are represented as nodes in a graph, while the containment relationships (e.g., a page containing a unit) and links (between pages or units) are represented as edges. To simplify the discussion, we assume that each node is annotated only with the name and the type of the concept it represents. In a real setting, model elements and relations can be annotated with all the properties specified in the language metamodel.

The graph representation is obtained exploiting the metamodel to create a model transformation that is applied to every project in the repository. Such a transformation can be expressed in any model transformation language (such as QVT or ATL). Considering our case study, the model depicted in Figure 2(a) can be transformed in a graph (encoded as a GraphML model²) as in Figure 3, where both a graphical (a) and XML (b) representations are shown. The whole project repository is transformed into a set of graphs offline, so to build the structured index required to perform queries. The query is processed exactly in the same way as the projects in the repository, thus obtaining graphs with equal representation, suitable for comparison.

4.3 Node Matching

The basic building block for any graph matching algorithm is *node matching*, i.e., the way in which one single node from a graph is matched to each node in the other graph. In our approach, we adopt a structured node matching approach, based on node distance. Node distance measures how similar are two nodes, considering one or more properties of the node itself. In our approach, the node matching operation is expressed as a function of the considered metamodel: given that a model element is described by its properties, node matching can be expressed as a multi-dimensional distance function that induces a topology (in a metric space) on the given nodes set. In our discussion, model elements are described by nodes annotated with a *name* and a *type* labels, and we express the distance function between two generic nodes n_1 and n_2 as follows:

² <http://graphml.graphdrawing.org/>

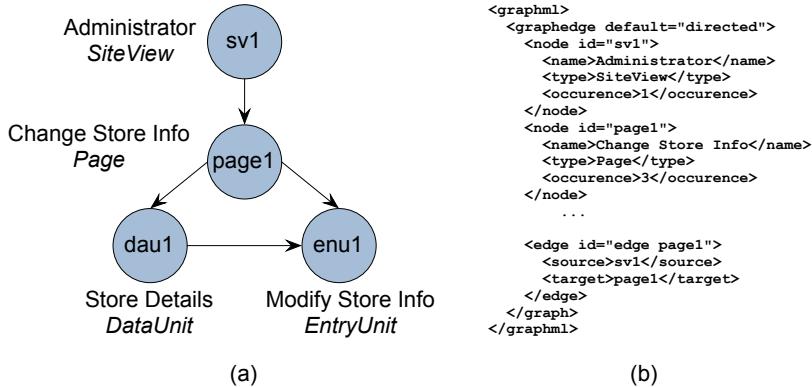


Fig. 3. Pictorial (a) and XML (b) representations of the graph used for indexing and search purposes for the WebML example of Figure 2

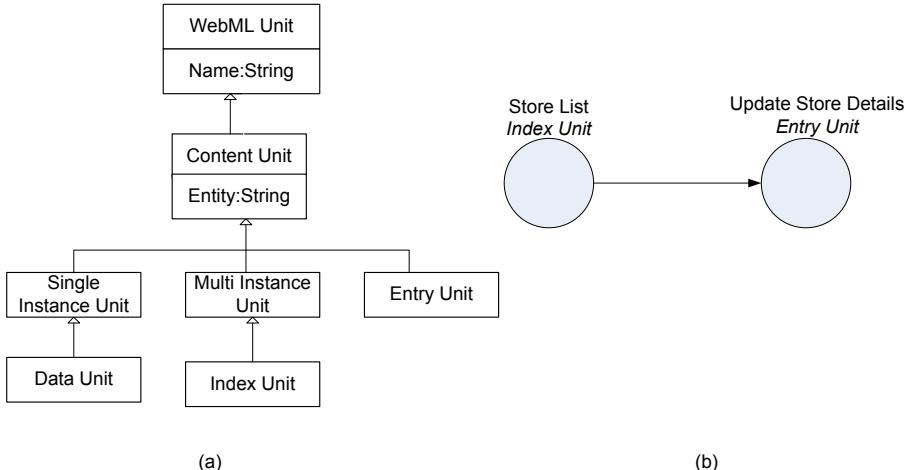


Fig. 4. (a) An extract of the WebML metamodel for element type comparison; (b) Graph representation of two WeML units

$$Dist(n_1, n_2) = \lambda \cdot levDist(name_{n_1}, name_{n_2}) + (1 - \lambda) \cdot typeDist(n_1, n_2) \quad (1)$$

where: *levDist* is the normalized Levenshtein distance (string-edit similarity normalized with the length of the longer string) applied to the node names; *typeDist* is defined based on semantic relationships between the compared types by exploiting the information contained in the project metamodel. In particular, it can be defined as the normalized node distance between the two classes in the metamodel graph.

The range of *Dist*, *levDist*, and *typeDist* is always $[0, 1]$. Given the above definition, $Dist(n_1, n_2)$ computes a standard Levenshtein distance when $\lambda = 1$, (thus

only considering the name of the involved nodes), and only the distance between model elements when $\lambda = 0$. For intermediate values of λ in the interval $[0, 1]$, a linear combination of both criteria is calculated. Figure 4(b) shows an example of two WebML elements for which the node distance calculation can be performed. From the example WebML metamodel of Figure 4(a) it can be calculated that the type distance between an *Index Unit* and an *Entry Unit* of Figure 4(b) is 0.75 (because the distance between the two classes is 3 and the maximum node distance in the graph is 4), while $levDist("StoreList", "UpdateStoreDetails")$ is 0.65. Therefore, with $\lambda = 0.5$ the distance between the two nodes is 0.7.

4.4 Graph Matching

Graph matching is the procedure of finding a mapping between two graphs for the purpose of calculating their similarity. Finding a complete node-to-node correspondence without violating both structure and label constraints of a graph is the problem of *graph isomorphism*. In most cases, graphs involved in the comparison have different dimensions. In our approach, the problem is related to *subgraph isomorphism*, which can be seen as a way to evaluate subgraph equality. A subgraph isomorphism is a weaker form of matching, as it requires only that an isomorphism holds between a query graph and a subgraph of the model graph [11].

Graph Edit Distance. In order to compare two graphs, a metric based on graph edit distance is used. The graph edit distance between two graphs is the minimal total cost of edit operations needed to transform one graph into the other [15]. The edit operations considered are:

- *Node substitution*: A node from one graph is substituted with a node from the other graph if they are similar;
- *Node insertion/deletion*: A node is inserted into a graph to match an existing one in the other or viceversa;
- *Edge insertion/deletion*: An edge is inserted into a graph to match an existing one in the other or viceversa.

Each of the above mentioned operations has a cost heuristically associated to it. The costs for insertion/deletion of nodes and edges are fixed to a value in the interval $[0, 1]$, while the cost for node substitution is defined as *1 minus the node similarity*. We also define a *cutoff threshold* on the similarity between nodes, so as nodes with similarity above threshold are considered similar and therefore substituted, while nodes below the threshold need to be inserted/deleted. In order to quantify how much two graphs are similar we normalize the graph edit distance to the range of $[0, 1]$. Then, the graph edit similarity between two graphs G_1 and G_2 is defined as [15]:

$$G_{Sim}(G_1, G_2) = 1 - \frac{w_{nI} \cdot f_{nI}(G_1, G_2) + w_{eI} \cdot f_{eI}(G_1, G_2) + w_{nS} \cdot f_{nS}(G_1, G_2)}{w_{nI} + w_{eI} + w_{nS}} \quad (2)$$

where f_{nI} , f_{eI} are the fractions of inserted nodes and of inserted edges respectively (i.e., the ratio between the inserted items and the total number of items), while f_{nS} is the average distance of the substituted nodes. The constant values w_{nI} , w_{nS} , and w_{eI} represent the weights for node insertion, node substitution, and edge insertion respectively, and can be varied to give more or less importance to each matching operation.

A-star Algorithm. The labeled graph representation of objects introduces noise and distortions. Moreover, often the query graph and the project graph do not have the same dimensions. Hence, it is necessary to include an error model and incorporate the concept of errors into graph matching. The graphs are then compared to each other by means of error-correcting subgraph isomorphism [11]. The most common approach to this problem is based on a search performed with the A-star algorithm on a tree representing the incrementally calculated *query to project distances*. A-star algorithm originally described by [5], is the best-first algorithm that finds the minimum cost path from one node to another in the distances tree. This algorithm is optimal since it examines the smallest number of nodes necessary to guarantee a minimum cost solution. The search space of the A-star algorithm can be greatly reduced by applying heuristic error estimation functions. By always expanding the node with the least cost, the algorithm is guaranteed to find the optimal mapping. Therefore, we adopt the A-star algorithm for error-correcting subgraph isomorphism detection that has been first applied in [11] and then adapted in [15]. The computational complexity of the algorithm depends on the size of the graphs, the number of labels and the number of errors [11].

The algorithm starts from a node n_1 in the query graph, and creates all the possible partial mappings from this node to every node in the project graph. Additionally, an extra mapping with a dummy node ϵ is created, (n_1, ϵ) , equivalent to the case where n_1 is deleted. The partial mapping with the maximal graph edit similarity is selected, and expanded into a number of larger mappings. The algorithm proceeds with the next node from the query graph, and creates partial mappings with every node from the project graph, excluding the ones already in the mapping. The algorithm always selects the mapping with maximal graph edit similarity, and expands it, by adding a mapping for the next node. The algorithm finishes when all the nodes from the query graph are mapped. To reduce the memory requirements, only mappings between similar nodes are allowed. In other words, only nodes whose distance defined by equation 1 is greater than a threshold parameter can form a partial mapping.

The algorithm that calculates the similarity between a query graph G_1 and a project graph G_2 is represented by the pseudocode shown in Algorithm 1, where N_1 , N_2 are the sets of nodes (and n_1 , n_2 are the nodes) of graphs G_1 , G_2 respectively; $open$ is the set of all allowed mappings, and map is the partial mapping having the maximal graph edit similarity $s(map)$. The algorithm finishes when all the nodes from the query graph are examined. The returned value is the maximal graph edit similarity for the two graphs.

Algorithm 1. A-star algorithm

```

Require:  $open \leftarrow (n_1, n_2) \mid n_2 \in N_2 \cup \{\epsilon\}, sim(n_1, n_2) > threshold \vee n_2 = \epsilon$  , for
some  $n_1 \in N_1$ 
while  $open \neq 0$  do
    select  $map \in open$ , such that  $s(map)$  is max
     $open \leftarrow open - map$ 
    if  $dom(map) = N_1$  then
        return  $s(map)$ 
    else
        select  $n_1 \in N_1$ , such that  $n_1 \notin dom(map)$ 
        for all  $n_2 \in N_2 \cup \{\epsilon\}$ , such that  $(n_2 \notin cod(map))$  and  $sim(n_1, n_2) > threshold$ 
        xor  $(n_2 = \epsilon)$  do
             $map' \leftarrow map \cup \{(n_1, n_2)\}$ 
             $open \leftarrow open \cup map'$ 
        end for
    end if
end while

```

Complexity analysis. Let us assume a query with n nodes and a project with m nodes. The complexity of the comparison algorithm [11] in the best case is $O(n^2m)$ (when all the nodes are uniquely labeled and the query graph contains an isomorphic copy of the model graph), while in the worst case is $O(nm^n)$ (when the error in the query graph is very large).

5 Experimental Evaluation

This section discusses how the graph-based model similarity search described in the previous section can be parameterized, and highlights the impact of the parameter tuning on the perceived quality of the search results. Our experiments were conducted on a project repository composed of 30 real-world WebML projects [1] from different application domains (e.g., trouble ticketing, human resource management, multimedia search engines, Web portals, etc.), all encoded as XML files conforming to the WebML DTD. Projects' size varied from 100 to 1700 nodes. We manually built a query set of 15 queries with different sizes for the evaluation of the approach: 5 small queries (1-25 nodes), 5 medium-size queries (26-450 nodes), and 5 large queries (451-680 nodes). The queries were chosen based on their size, and different structure coverage.

We performed a manual assessment of the query-to-project relevance, so to establish a ground truth for performance evaluation. The assessment was carried out by one WebML expert analyst with modeling and application proficiency. Each query was manually compared against the project repository using the WebML models XML representation, considering element names, element types, and hierarchical positioning of the elements in the query and the project. Then, a relevance for each query against every project has been assigned, ranging from 0 – 10 (with steps of 0.5), where 0 implies no relevance (i.e. the query has no match in the project), and 10 indicates maximal relevance. This evaluation

Table 1. Experimental parameters and Spearman’s coefficient (mean and std on 15 queries) for experiments with node type similarity

λ		Exp.1	Exp.2	Exp.3
	$w_{edgeIns}$	0.1	0.1	1.0
	$w_{nodeIns}$	0.1	1.0	0.1
	$w_{nodeSub}$	1.0	0.1	0.1
0.25	mean(ρ)	0.66	0.40	0.52
	std(ρ)	0.11	0.16	0.18
0.5	mean(ρ)	0.65	0.50	0.54
	std(ρ)	0.11	0.18	0.17
0.75	mean(ρ)	0.67	0.64	0.63
	std(ρ)	0.12	0.15	0.16

induced a ranking of projects with respect to each query. Such ranking has been used as a reference ground truth for the evaluation of the algorithm.

To assess the robustness of the approach with respect to parameter changes, and to identify the optimal set of parameter values for the considered DSL, we tested our approach with different configurations of *node distance*, *node insertion*, *node substitution*, and *edge insertion* weights. Our analysis focused on two aspects: the quality of the results and the performance of the algorithm. The result quality is evaluated with respect to different parameterizations of the algorithm, while the performance is independent on the parameters, and thus is reported for one scenario only.

5.1 Analysis of the Quality of the Results

Our aim is to evaluate the impact of the parameters of the algorithm on the results quality. According to the definition of the algorithm, we can tune two types of parameters: the weights w_{nI} , w_{nS} and w_{eI} assigned to the frequencies of the graph edit distance (Eq. 2), and the value of the parameter λ in the node distance function (Eq. 1). In our experiment we defined 3 different configurations of these weights, and three different values of λ . The weight configurations have been selected in order to highlight the contribution of each component in the graph distance calculation. The values of λ have been set to respectively 0.25, 0.5 and 0.75. Overall, we obtained 9 different experimental scenarios. The node type similarity cutoff threshold has been set to 0.6.

For the evaluation we performed the 15 queries in all the 9 scenarios and we calculated the average values for the quality measures. Each scenario has been evaluated considering the quality of the produced rankings (w.r.t. the manual assessment) according to the Spearman’s Rank Correlation Coefficient and the Discounted Cumulative Gain (DCG).

The **Spearman’s Rank Correlation Coefficient** [17] is a non-parametric correlation coefficient typically used in information retrieval to measure the correlation between ranks (i.e., synthetic linear relationships imposed over non-linearly associated variables), as:

Table 2. Spearman’s coefficient (mean and std on 3 experiments) for the 15 queries

λ	Size	Small queries					Medium queries					Large queries				
		Query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
0.25	$mean(\rho)$	0.58	0.71	0.51	0.76	0.49	0.48	0.56	0.34	0.47	0.44	0.60	0.63	0.58	0.33	0.45
	$std(\rho)$	0.10	0.12	0.06	0.01	0.10	0.14	0.04	0.05	0.22	0.28	0.09	0.14	0.18	0.31	0.29
0.5	$mean(\rho)$	0.60	0.78	0.57	0.77	0.53	0.51	0.57	0.38	0.51	0.43	0.62	0.69	0.62	0.38	0.46
	$std(\rho)$	0.08	0.00	0.04	0.00	0.09	0.11	0.07	0.19	0.15	0.25	0.05	0.04	0.07	0.22	0.24
0.75	$mean(\rho)$	0.64	0.78	0.58	0.81	0.59	0.57	0.66	0.39	0.65	0.72	0.64	0.72	0.69	0.47	0.81
	$std(\rho)$	0.01	0.00	0.04	0.00	0.02	0.00	0.01	0.21	0.06	0.14	0.05	0.05	0.10	0.24	0.05

$$\rho = 1 - 6 \cdot \frac{\sum_{i=1}^n d_i^2}{n \cdot (n^2 - 1)} \quad (3)$$

where d_i is the difference (in terms of number of positions) of two items in the two compared ranks. It has values in the range of [-1,1], where: -1 corresponds to perfect negative correlation, 1 to perfect positive correlation, and 0 represents no correlation between the variables. In our case, we calculate the correlation between the results produced by our algorithm and the ground truth built manually. The more these are correlated, the best is the quality of the algorithm. Hence, values closer to 1 demonstrate better performances.

The **Discounted Cumulative Gain (DCG)** [6] is a well-known metrics for assessing the relevance of search result sets, which measures the usefulness (gain) of a document based on its relevance w.r.t. the query and its position in the result list. DCG computes the cumulated gain by introducing a logarithmic discount factor based on the rank position i of the retrieved item as follows:

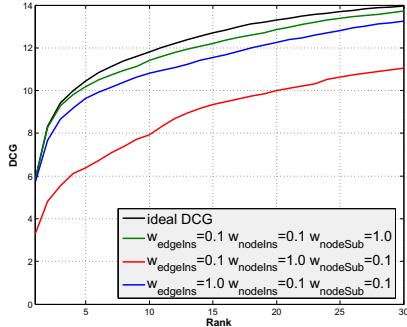
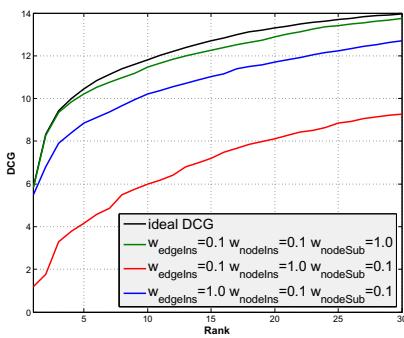
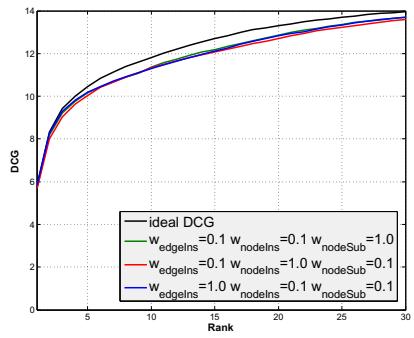
$$DCG_p = \sum_{i=1}^p \frac{2^{rel_i} - 1}{\log_2(1 + i)} \quad (4)$$

where rel_i is the relevance for the i -th rank position.

Table 1 shows the different parameter combinations and the Spearman’s coefficient results. The table shows that the mean Spearman’s correlation coefficient between the ground truth and the calculated results is good for all the experiment configurations (with low standard deviation), thus showing good retrieval performance. For all three experiment configurations, the best performance is obtained when the maximal weight is given for node substitution. As λ increases, the difference in the results among the experiments decreases.

Table 2 delves into the detailed behaviour of the 15 different queries (averaged on the 3 experiments). As λ increases, the performance of the algorithm improves, since the mean value for the Spearman’s correlation coefficient increases, and the standard deviation decreases, independently from the query size.

These results are compatible with the DCG analysis, that has been performed to evaluate the perceivable quality of the result sets. Figure 5 shows the behaviour of DCG in the ideal manual assessment and the ones obtained by our

(a) $\lambda = 0.5$ (b) $\lambda = 0.25$ (c) $\lambda = 0.75$ **Fig. 5.** DCG for $\lambda = 0.5$ (a) $\lambda = 0.25$ (b) and $\lambda = 0.75$ (c)

algorithm depending on the different weight values. The algorithm generally performs well, since the values for all the experiment settings are close to the ideal DCG curve corresponding to the manual assessment. This means that the algorithm is able to retrieve relevant projects at highly ranked positions.

The best result is obtained with $w_{nodeSub} = 1$, i.e., when the substituted nodes are accounted as more important in the overall similarity comparison; this is aligned with the intuition that similar (i.e., substituted) nodes are more significant than inserted nodes and edges in the calculation of the graph similarity. Inserted edges ($w_{edgeIns} = 1$) are deemed as slightly less significant, while inserted nodes ($w_{nodeIns} = 1$) behave even worse.

The DCG analysis on λ assessed that higher values of λ imply that the DCG curves tend to get closer, as shown in Figure 5. This means that the role of the three weights becomes less and less important when λ increases. Once more, this is intuitive since higher λ values represent the fact that text distance between nodes is assumed as more relevant while type distance (and the three weights together with it) are considered less important. In any case, the experiment with $w_{nodeSub} = 1$ is consistently the best one independently on the value of λ .

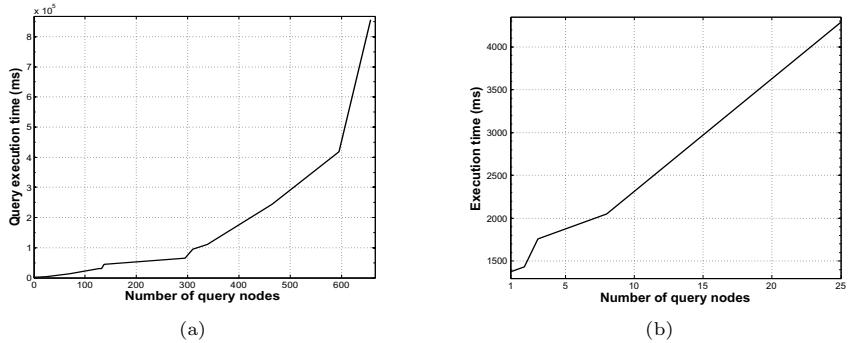


Fig. 6. Query execution time w.r.t. query size(a) and detail for small queries(b)

5.2 Performance Analysis

Figure 6 (a) shows the query execution time for all the 15 queries considered in the experiments with respect to the query size, i.e. number of nodes in the query. The image highlights the exponential behavior of the algorithm, as expected given the complexity analysis in Section 4. However, while this is interesting from a theoretical viewpoint, the exponential complexity is not so crucial when dealing with queries of reasonable size (that can be defined in terms up to 50 nodes or so). Indeed, the behavior in this scenario can be assimilated to linear, as Figure 6 (b) shows, granting results query execution time in the order of 1 to 5 seconds. Notice that no query execution optimization (including optimized indexing of the repository) has been adopted during experiments and therefore we foresee a wide range of possibility for improving the performance of the system.

6 Conclusions

In this paper we presented a graph-based approach for content-based search of models using the A* algorithm. We evaluated our approach upon a realistic setting, obtaining good quality of results with respect to a manually assessed ground truth, together with a clear profiling of the parameters behavior.

Future work includes: increasing the size of the project repository and of the ground truth; comparing our graph search solution with keyword based ones [1]; studying techniques for automatically tuning the parameters of the A* algorithm, such as e.g. neural networks or genetic algorithms ; and experimenting the approach with other metamodels, such as e.g. BPMN.

References

1. Bozzon, A., Brambilla, M., Fraternali, P.: Searching Repositories of Web Application Models. In: International Conference on Web Engineering, pp. 1–15 (2010)
2. Brügger, A., Bunke, H., Dickinson, P., Riesen, K.: Generalized Graph Matching for Data Mining and Information Retrieval. Advances in Data Mining. Medical Applications, E-Commerce, Marketing, and Theoretical Aspects, 298–312

3. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Morgan Kaufmann series in data management systems: Designing data-intensive Web applications. Morgan Kaufmann Pub., San Francisco (2003)
4. Dijkman, R.M., Dumas, M., van Dongen, B.F., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. *Inf. Syst.* 36(2), 498–516 (2011)
5. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2), 100–107 (1968)
6. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20(4), 422–446 (2002)
7. Kunze, M., Weske, M.: Metric trees for efficient similarity search in large process model repositories. In: Proceedings of the 1st International Workshop Process in the Large (IW-PL 2010), Hoboken, NJ (September 2010)
8. Lucrédio, D., Fortes, R.d.M., Whittle, J.: MOOGLE: A model search engine. *Model Driven Engineering Languages and Systems*, 296–310 (2010)
9. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic schema matching with cupid. In: Proceedings of the International Conference on Very Large Data Bases, Citeseer, pp. 49–58 (2001)
10. Markovic, I., Pereira, A.C., Stojanovic, N.: A framework for querying in business process modelling. In: Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI), München, Germany (2008)
11. Messmer, B.: Efficient Graph Matching Algorithms for Preprocessed Model Graphs. PhD thesis, University of Bern, Switzerland (1996)
12. Nandi, A., Bernstein, P.A.: HAMSTER: using search clicklogs for schema and taxonomy matching. In: Proceedings of the VLDB Endowment, vol. 2(1), pp. 181–192 (2009)
13. Nowick, E.A., Eskridge, K.M., Travnicek, D.A., Chen, X., Li, J.: A model search engine based on cluster analysis of user search terms. *Library Philosophy and Practice* 7(2) (2005)
14. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. *The VLDB Journal* 10(4), 334–350 (2001)
15. Remco Dijkman, M.D., Garcia-Banuelos, L.: Graph matching algorithms for business process model similarity search. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 48–63. Springer, Heidelberg (2009)
16. Smith, K., Bonaceto, C., Wolf, C., Yost, B., Morse, M., Mork, P., Burdick, D.: Exploring schema similarity at multiple resolutions. In: Proceedings of the 2010 International Conference on Management of Data, pp. 1179–1182. ACM, New York (2010)
17. Spearman, C.: The proof and measurement of association between two things. *The American Journal of Psychology* 100(3-4), 441 (1904)
18. Syeda-Mahmood, T., Shah, G., Akkiraju, R., Ivan, A.A., Goodwin, R.: Searching service repositories by combining semantic and ontological matching
19. Tian, Y., Patel, J.M.: Tale: A tool for approximate large graph matching. In: International Conference on Data Engineering, pp. 963–972 (2008)

AdapForms: A Framework for Creating and Validating Adaptive Forms

Morten Bohøj¹, Niels Olof Bouvin², and Henrik Gammelmark³

¹ Alexandra Institute, Åbogade 34, DK-8200 Aarhus N

² Dept. of Computer Science, Aarhus University, Åbogade 34, DK-8200 Aarhus N

³ Dat1, Åbogade 15, DK-8200 Aarhus N

Abstract. AdapForms is a framework for adaptive forms, consisting of a form definition language designating structure and constraints upon acceptable input, and a software architecture that continuously validates and adapts the form presented to the user. The validation is performed server-side, which enables the use of complex business logic without duplicate code. Thus, the state of the form is kept persistently at the server, and the system ensures that all submitted forms are valid and type safe.

1 Introduction

The World Wide Web has gained its remarkable success not only because it is a (relatively) simple and scalable publishing system, but also because it offers easy access for users to add new content or interact in other ways by filling on-line forms. This is done many times daily in search fields, e-shops, application forms, and so on. While many of these text fields and forms are straightforward, some are more challenging for the user to correctly interpret and fill. Likewise, an essential part of form handling at the developer's end is the checking and validation of the input, before it can be added to a database or used as basis for other calculations. These validation steps range from trivial tests that can be evaluated using e.g., string length over more complex rules checked with regular expressions, and finally input that can only be checked against the “business logic” of the particular application. In practical terms, this can range from checking whether a family name has been entered, whether an amount entered consists only of digits, or whether a desired vacation period is actually available. As many administrative processes move on-line, the associated forms follow. Forms may be simple or complex with opaque and non-trivial interrelationships that can be difficult to communicate effectively to the user. When is a form filled out correctly or sufficiently; and how do the choices in one field affect the rest of the form? Which parts of the form are relevant, and what can be safely ignored?

We describe in this paper AdapForms, a general solution to handling forms in a fashion that enables developers to clearly designate accepted types of input, reuse existing templates (e.g., sub-forms for postal addresses), use complex validation rules, allow users to resume filling out complex forms at a later point, and communicate the state of the form to the user unobtrusively. An open source

demonstrator of the framework is available online¹ and an exhaustive description can be found in [6].

The paper is structured as follows: Our system is described from a functional point of view in Section 2 and an architectural one in Section 3, the system has been evaluated as described in Section 4, we describe related work in Section 5, and the paper is concluded in Section 6.

2 The AdapForms Framework

Validation of user submitted data on the Web is usually a two-step process—input is validated in the browser (usually through custom JavaScript code), and validated again when submitted to the server. Client-side validation has certain advantages, especially responsiveness as no network latency is involved. However, client-side validation cannot replace server-side validation, as the input sent back to the server may be subverted or corrupted accidentally or maliciously. Thus, server-side validation is still necessary—at the very least to thwart SQL injections and similar attacks. AdapForms is a server-centric framework, which

(a) Yellow and red signify missing resp. invalid data. The Parent form is due to the age of the user

(b) The applicant is of legal age, and thus Civil status is relevant

Fig. 1. AdapForms screenshots

allows direct code access to the form and data structures, described further in Section 3. This also means that all validation is done on the server instead of the client, which provides a higher degree of safety regarding validity of the input. An advantage of having the state constantly up-to-date server-side, is that it allows direct transparent integration with the host application instead of relying on additional server calls for adaptation and validation, as described below. Furthermore, business logic is not exposed in JavaScript accessible in the browser, but is kept on the server.

¹ <http://adapforms.gammelmark.eu/>

While the AdapForms framework is focused on the underlying handling of form structure and data and not on the user interface to the form, an example XHTML/Ajax based prototype implementation of the UI has been created for testing and demonstration purposes. This example UI could also have been created using HTML5 and taking advantage of the new form elements. HTML 5 thus provides a richer user experience without replacing the AdapForms framework. The web UI is used as example throughout this paper and two screen shots are shown in Figure 1(a) and 1(b). The screen shots (discussed in more detail in Section 2.2) illustrate how forms are validated and adapted to the user's input. The UI is not part of the framework and UIs for multiple platforms could be created, without changing the document XML definition or server side validation. Implementing different UIs also allows for changing the rendering of how feedback is given to the users.

Listing 1.1. Sample XML form

```

1 <adapforms>
2   <include file="templates.xml" />
3   <form title="Example Form">
4     <defaults readonly="false" write-roles="applicant" />
5     <template name="anAddress">
6       <group>
7         <text id="street" label="Street">
8           <validator criteria="lower-case(.) = ."
9             message="Must be all lower-case" />
10        </text>
11        <text id="city" label="City" pattern="[a-z]*" />
12        <integer id="zip" label="ZIP/postal code" />
13      </group>
14    </template>
15    <group id="info" label="Personal information">
16      <use template="anAddress" id="address" label="Current address" />
17      <integer id="age" label="Age" minValue="0" />
18      <toggle id="parental" label="Parental accept" relevant=".../age &lt;= 18"
19        write-roles="parents" />
20      <text id="parentName" label="Name of parent" relevant=".../parental/@relevant" />
21    </group>
22    <bean id="payment" label="Payment details" type="myapp.model.PaymentInfo" />
23    <repeat id="kids" label="Children" entryLabel="Child" minRepeats="0" />
24      <text id="name" label="Childs name" />
25    </repeat>
26  </form>
27 </adapforms>
```

2.1 Defining Forms

The form structure in AdapForms is defined in a XML document, which is parsed by the host application and forms the basis for the internal representation of the form. An example is seen in Listing 1.1. This sample is meant to demonstrate features rather than show a typical form. The XML form can contain the following elements representing different types of data:

Label Read-only text label.

Text Regular text element, single- or multi-line.

Secret Text field for masked input.

Integer & Decimal Ranged or unranged integer or decimal respectively

Toggle True or false.

Choice Single selection among fixed, mutually exclusive, choices.

Multi-choice Multiple selections among fixed choices.

Date Single date selection.

HelpText Text to help the user fill out the form.

Group Logical grouping of elements in the form. **Group** can contain the elements described above and group them together. This grouping allows for an easier overview of the form and allows for making an entire section e.g., **relevant** or **required**. An example of the **group** element is shown in line 14 of Listing 1.1.

Repeat Repeats are a collection of form elements that can be repeated in a list-like manner. This can e.g., be relevant for allowing users to add a variable number of children, without knowing how many elements to create beforehand. The number of repeats allow can be limited or unlimited and both as a required minimum or limited maximum. A small example of a **repeat** element is shown in line 22 of Listing 1.1.

Bean A bean is a **group** generated from a JavaBean—see below.

Each of these form elements has a number of possible attributes. Some of these attributes are specific to a particular form element, while others are shared among all elements. Most are optional and only **id** and **label** are required. The common attribute set is as follows:

id A locally unique ID. The ID must be unique among siblings in the XML tree. The ID is required for all form elements.

label A textual label explaining the element's purpose. The label attribute is optional for the *group* and *bean* elements and required for the rest.

relevant Boolean value indicating the relevance for the user. Relevance can e.g., be used to hide irrelevant elements from the user.

required Indicates whether or not the element is required.

read-roles & write-roles List of user roles allowed to read or write, respectively, the specific element. The user roles are handled by the host application when loading the form. The use of roles is not required. User roles may be used to allow more rights to some users, such as administrators.

readonly Indicates that no user is allowed to change the element value.

uiflags Text string which may hold UI specific information.

If an optional attribute is not present in an element, the value is inherited from the parent node. If no parent node is present, the value is taken from the nearest **defaults** element (see line 4 in Listing 1.1). The sole purpose of the **defaults** element is to provide default values, and if no default value is found, the framework defaults apply. When designing forms, one might experience defining the same element structure more than once. This could e.g., be elements of an address section. To allow for reuse of such reoccurring structures, the framework has a template option. Templates can be defined within the same XML document as the rest of the form, as in line 5–13 of Listing 1.1, with the template used in line 15, or they can be placed in a separate XML document and included in the form using the **include** element, as in line 2 in Listing 1.1. This supports the reuse of common structures, such as addresses, across forms. Similar to the

use of templates, the framework allows for the use of JavaBeans. A bean is included in the form using the `bean` element as shown in line 21 of Listing 1.1. The use of JavaBeans results in a group node in the form structure, which contains the public available properties. The framework supports properties with the following Java types: `String` (Represented as `Text` element), `int` or `long` (`Integer` element), `float` or `double` (`Decimal` element), `boolean` (`Toggle` element), `java.util.Date` (`Date` element) and `enum` types (`Choice` element). The framework does not currently support beans within beans. The individual properties can be annotated in the Java class using `@AdapFormsProperty` to support e.g., renaming the label or designating required attributes. Annotating the bean class itself defines defaults for all properties.

2.2 Adaptation

One of the core features of AdapForms is the ability to adapt the form according to user input. These adaptations could be changing/setting element values based on input in other fields, or hiding/showing parts of the form that may be irrelevant/relevant based on user input. An example of the latter could be to hide the workplace address field, if the user indicates she is currently unemployed. Figure 1(a) and Figure 1(b) shows an example of this adaptation, where in Figure 1(a) the date of birth is set to 2009, thus giving an age under 18 and therefore requiring a parental section. In Figure 1(b), the age is above 18 and the parental section is replaced by the question of marital status. By hiding fields not relevant to the user, we eliminate some potential confusion for the user, so she can concentrate on the fields that are relevant to her. Likewise, by filling out some fields based on either entered information or previous knowledge, we minimise the typing needed by the user, as she can merely verify the information.

XPath. One way to perform form adaptation is to do it directly in the XML document using AdapForms' integration of XPath 2.0[1] expressions. An example of the use of XPath expressions is shown in line 17 of Listing 1.1. Here the relevance of the toggle element `parental` is dependent on the value of the sibling element `age` being less than or equal to 18. XPath is very expressive and has the advantage of placing the validation rules close to the adaptation targets. As XPath is often used in connection with XML, the developer is also more likely to be familiar with it than having to learn a completely new language. While XPath is powerful, it has some limitations in this setting, e.g., it can only affect a single state parameter as it only has one output value. It can also only look at the information typed in the form and not access previously known information.

Form Hooks. The adaptations mentioned above are pre-coded adaptations meaning that they are based on pre-coded rules and anything learnt from other users filling out the same form can not be taken into account. This is where the AdapForms goes beyond using XForms and XPath. AdapForms supports more complex adaptations through the use of form hooks. A hook is defined

by the Java interface `FormHook` which defines three methods `onValueChange`, `onRepeatEntryAdd` and `onRepeatEntryRemove`. Each hook is registered with a specific form path, described in Section 3.2, before the form instance is initialised. The same hook may be registered with more than one form path or no path, if, for some reason, the developer wants a single hook to receive all change events generated. The hook is notified through the aforementioned methods, when a change occurs at the registered path. `onValueChange` is called on value changes, whereas `onRepeatEntryAdd` and `onRepeatEntryRemove` is called for changes on repeat entries. Changes on form-level, such as submission of the form, is handled through the `InstanceCallback` interface, and here only a single callback can be registered to a form instance. Because hooks are created using Java, it allows for more options with regards to adaptation, as a hook can manipulate several values or look up previously entered information in a database. Hooks also make it possible to base the adaptations on arbitrary business logic, or more complex computations using machine learning, if these solutions are Java based. Machine learning would e.g., allow for the form to change a default choice in a multiple choice to reflect the choice of the majority of users.

2.3 Validation

When receiving data through web forms, it is essential to validate the input prior to processing. This validation can be anything from checking whether a field is a valid date to checking that a supplied user name is known. Often frameworks, such as PowerForms[4], perform this validation client-side in order to provide quick user feedback, but this means that the same validation must be done at the server as well to make sure no invalid data slips through. AdapForms performs the validation server-side, while continuously providing feedback to the client. This adds some time for validation, but ensures that the form instance on the server holds the correct information. The validation is done asynchronously, so the latency is usually not noticed.

The framework operates with three different kinds of problem definitions: `Error`, `Warning` or `Required`. `Error` indicates that the input is invalid. This could be entering a string where a number is expected. `Warning` indicates that while the input is not invalid it may be problematic. This could e.g., be if the user have put the year 1010, where it is more likely that he meant 2010. `Required` indicates that an element that is required is missing some input. These problem classifications lead to a form instance being able to be in the following states:

Uninitialised The form instance has been created, but is not yet initialised and can thus not handle input

Initialising The initialising process is ongoing

Invalid The form contains one or more problems. In Figure 1(a) this is highlighted by displaying an error message next to the submission button.

Incomplete The form is valid except for errors of the `Required` kind. Figure 1(b) shows how this could be shown to the user by a warning message next to the submission button.

IncompleteWithWarnings The form may hold **Required** or **Warning** errors

ValidWithWarnings The form only holds errors of the **Warning** type.

Valid The form holds no errors

On submission of the form, the host application may choose to accept or reject the form based on the form status. The host application can use the form status to perform different actions based on the status, e.g., saving incomplete forms for later completion instead of submitting it.

The validation in AdapForms takes place on three different levels: basic framework validation, validation rules, and host application. The final form status is based on the union of the errors on these three levels.

The basic framework validation includes:

Required field validation Checking that required fields contain a value. This could be shown as in both Figure 1(a) and Figure 1(b) with yellow warnings.

Type check Checks that e.g., integer elements only holds integers or date elements contains only dates.

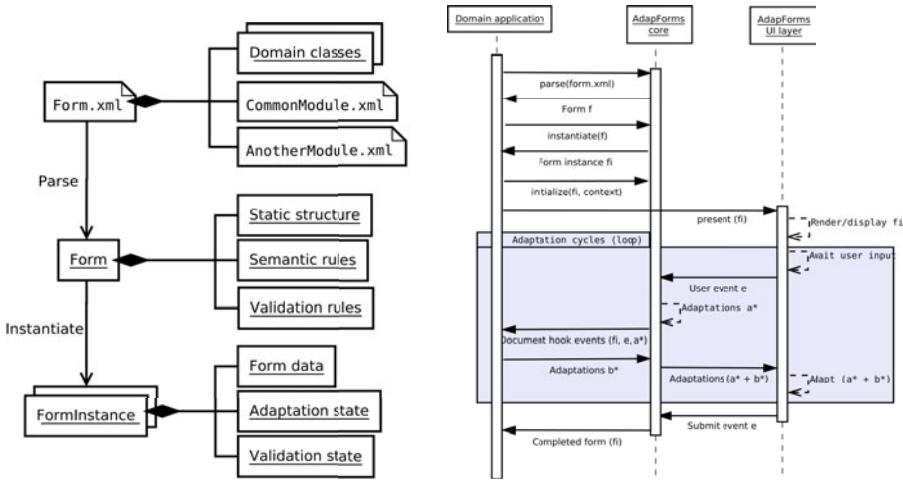
Range check Integer elements may be upper- or lower-bounded, as shown in line 16 of Listing 1.1, as can the length of text elements

Format check Text elements may define regular expressions the input must uphold as in line 10 of Listing 1.1

This built-in validation will trigger when the form is loaded and when values are changed. The host application may not override these mechanisms to avoid inconsistencies. The XML form itself may also contain more complex validation rules by using XPath expressions. These can be placed in **validator** elements, which is then placed as children of the element it validates. Each **validator** element has a **criteria**, which contains an XPath expression, and a message to give the user in case of an error. An example of a validator is shown in line 8 of Listing 1.1, where it checks that the street is written only in lower-case letters. Validators are mostly used to validate values, but are not limited to this. Validators are evaluated on every change of values in the form, and not just on change of the element the validator is attached to. As a third option of validation, the host application may set or remove validation rules for individual form elements. This is most easily done using **FormHooks**, attached to the element that needs checking. Hooks would typically be used to perform more elaborate validation that is not possible with the built-in validation, such as comparing input to values in a database. The host application may decide to perform validation any time and is not limited to the use of hooks.

3 Architecture

The framework is built using the Model-View-Controller pattern, where the core framework essentially comprises the Model and Controller by providing the data structures as well as the logic making the model dynamic. The View part is UI-specific and thus highly coupled to the display- and interaction-capabilities

**Fig. 2.** AdaptForm in action

of the technology chosen. How to specifically display data to, and interact with the user, is not the primary concern of the framework. Instead the focus is on determining what to display and what to do with the input gathered. An important goal of the architecture is to have the host application manipulate the form and interact with the user exclusively via local proxy interfaces, regardless of how the UI exposes the form and where it is physically being executed.

3.1 Form Life Cycle

The software architecture of AdapForms can best be understood by considering the life cycle of an adaptive form, so that the various phases can be related to concrete software elements. The Instantiation and Initialisation phases can be thought of as a single phase in the abstract sense.

Definition. The form syntax and semantics are defined in a XML file external to the framework.

Parsing and loading. The form is parsed and its dependent definitions and objects are parsed, ending up in a single **Form** entity representing the static form, with all its form elements (explicitly specified, or inferred from templates, beans etc.). The form structure may be viewed as an abstract structured tree, where each node is a form element. See Figure 2(a) for a visual illustration of the process.

Instantiation. A concrete instance of the form is created. Using the static form as a basis, the instance will hold current values, validation status etc. of all nodes in the static form structure. The **FormInstance** is essentially holding the collective state of the instantiated form. Multiple users can share the same **Form**, but they will each have a unique **FormInstance**.

Initialisation. The form instance is initialised with default values and, if available, stored form data from a previous session. The role of the current user (if any) is also selected at this point. All relevant semantic rules (i.e., XPath and hooks) are triggered, causing the form to be pre-adapted as much as possible before the user encounters the form.

User interaction. Input from the user (or the host application) is passed to the instantiated form, and the instance may respond with a list of validation errors and form adaptations, triggered by the loaded semantics. Each user interaction triggers a new adaptation cycle.

Adaptations may be scripted with XPath expressions and queries. To facilitate complex domain-specific validation and reasoning, the host application may register hooks with the instance, which are triggered whenever a given form element changes its value or state.

Submission. At submission, the instance is checked for any consistency problems and validation errors, and the event is then passed (along with the instance) to a callback handler, specified by the host application.

3.2 Form Paths

A recurring scheme is the use of *form paths*. A path uniquely identifies a form element, and is used throughout the framework for multiple purposes. Examples include look-up of form elements (form structure tree), addressing nodes in the internal state tree, defining semantic rules etc. A path resembles the hierarchical file names, and is heavily inspired by XPath and by h-maps [8]. An example form path looks like this: /applicant/spouse/name/first. Paths may be fully qualified (indicated by a leading slash) or relative to the context they are used in. For the most part it is possible to create a 1-to-1 mapping between a form element in the static form with the representation of that field in the instantiated form, simplifying things. However, this is not possible in the case of repeated sub-forms, where the user may specify any number of occurrences of a given data type. In these cases, an extra annotation level to the instance paths is introduced, so the paths become /applicant/jobs[1]/company and /applicant/jobs[2]/company for the first and second repeat, respectively. This syntax is also similar to XPath. When making references to the static form definition, the numbering scheme is omitted. When defining semantic rules, both versions may be used, depending on whether it is a general rule, or a rule specific to a single repeated entry.

3.3 Element State Tree

The form instance holds state for each form element present in the instance. The actual state is held in a hierarchical data structure with form paths as keys.

This is also inspired by how FormGen [5] stores state, but with the major difference that the host application need not be implemented against specifically generated tree classes. Thus, any host application can load any adaptive form and process it at run-time, although it may of course not always be able to do

anything sensible with the entered data besides saving or forwarding it. The trade-off of this approach is that the API exposed to the host application must be more general and therefore may be less natural to use, than if the actual tree was generated specifically for the domain.

At any given time, the following state parameters are available for all elements: The unique form path and a **Relevant** state parameter, specifying whether the element is currently visible to the user. In addition, any form element capable of holding a value, also holds the following state: Current value, **ReadOnly** state parameter, **Required** state parameter (for validation), list of associated hooks, and the validation state. In Figure 2(a), the state tree is represented by the Form data, Adaptation state, and Validation state collectively.

All of the mentioned state entries can be read and manipulated by the host application, with the exceptions stated above. Changing the state will in most cases trigger an adaptation of the form instance. Access to reading and manipulating the state of a given element is exposed by the **ElementState** interface, allowing direct API manipulation.

As mentioned above, the tree is partially inspired by hierarchical maps [8]; a data structure intended for multi-purpose storage of data and state of varying complexity using an ad-hoc tree-like structure, that changes over time. Each piece of information is stored in its own leaf node, and is identified by a unique path, where each part of the path represents a level in the tree, starting from the root. In addition to holding values, a leaf may have registered one or more message handlers, which are invoked in order, when a message is destined for the corresponding tree path. Due to the tree structure, serialisation to and from XML is straightforward.

In contrast to the original h-map, the set of possible nodes is restricted to match the paths dictated by the static form structure, with the exception of **Repeats**, as indicated above. Thus, every update to the structure is checked against the form structure. This serves several purposes:

- Type safety is guaranteed, as only data of the type dictated by the corresponding form element can be written. When retrieving data from the structure, there is thus no need for type checking, and any type errors are caught early. Automatic data type conversion is applied, where applicable.
- Upon querying or updating a value in the structure, the action is checked against the role permissions of the element. This guarantees that no data is accidentally overwritten by a user not having the correct role.
- It is guaranteed that no “phantom” values are inserted, that are not logically linked to the form due to path typos, logic errors, or similar.

Although most of this should already be implicitly guaranteed by the UI layer implementation, it serves as a final centralised sanity check and guards against both programming errors and attempts to craft malicious input. Furthermore, the host application may manipulate parts of the data structure directly, and once again this therefore guarantees a sound state of the structure at all times.

While each state tree node has its own set of parameters, these may be overruled at run-time due to the semantic inheritance. Consider the case of two

nested form elements; the logical group `/foo` and its contained text element `/foo/bar`. If the `relevant` flag of the former element is changed to `false` this value will propagate to the latter as well.

Mechanisms are provided for the form semantics logic to query and modify both the stored (possibly overruled) and the active value, but regardless of how the parameters are manipulated, the inheritance overrule remains in effect. Inheritance rules are out of scope for this article, but are described in [6].

Adaptation Cycles. All adaptations within a form instance are initiated by a user action. This means that the form cannot suddenly change in front of the user, unless he or she has triggered the adaptation by changing one or more values in the form, submitting it, or have otherwise actively performed an action.

The term adaptation cycle denotes a complete cycle from the reception of user input (value change, or submission request), and until the user experiences the adaptations. Although the adaptation cycle is always triggered by a single user action, the triggering of form hooks or semantic rules may cause a cascade of adaptations to take place before the cycle ends. The host application can perform changes to the form instance at any given time. However, these adaptations will be accumulated in a buffer, and will only be presented to the user at the next adaptation cycle to avoid confusion.

An adaptation is conceptually anything from a value change, validation status change, relevance change, read-only status change to more volatile events such as displaying a message to the user and more UI-specific adaptations, such as redirecting a browser user to another website, closing a window, or playing audio.

Figure 2(b) illustrates the interaction sequences and the role of the iterative adaptation cycles within a control flow context. Each iteration of the outer loop represents a complete adaptation cycle. A single user-initiated adaptation may trigger further adaptations and form hooks.

UI-specific View Component. The UI layer responsible for interfacing between the user and the core framework is only defined abstractly in the general architecture, as a component with the following primary responsibilities:

- Presenting a pre-adapted form instance to the user.
- Populate the form elements with data and sending it to the core component.
- Adapting the presented form, as indicated by the core component.
- Highlighting validation errors within the presented form, based on feedback from the core component.
- Submitting the form, when the user deems it to be completed.

Note, that we are not restricted to graphical interfaces. Any interface, e.g., audible or tactile, could theoretically be implemented, although the mapping is not necessarily straightforward. A proof-of-concept implementation for a XHTML/Ajax web platform has been implemented, as mentioned previously.

4 Evaluation

A framework such as AdapForms must be viewed from at least three perspectives, when considering its fitness as a tool, namely the perspectives of the developer (creating the forms), the form filler (the user filling in information), and the form handler (responsible for handling the input). We have conducted some initial user feedback studies on each of the three perspectives.

Form Filler Feedback. As mentioned above, a prototype user interface for AdapForms has been implemented. The form filler feedback was conducted on a version prior to the demonstrator web UI included in the current framework version. Changes were only made to the demonstrator UI and not to the handling framework. The screen shots in Figure 1(a) and 1(b) are from the new version. In the version used for the test, the framework indicated correctly filled fields with green icons, and indicated errors and warnings with red and yellow icons respectively. Error and warning messages were only available when hovering over the icons next to the field.

In order to get feedback from form fillers, the web UI prototype was used for experimentation. Three users were presented with a form produced with the AdapForms framework, using the example UI, and a similar form implemented in standard static HTML. In order to do a reasonable comparison, the static form performed the same validation on submission as the adaptive form did during completion and gave the same messages at the top of the document.

One of the major issues discovered with the adaptive form was with the validation feedback given to the user. The coloured icons indicating validation status next to the fields were largely ignored by the users, and the users were puzzled as to why they could not submit the form. When the users noticed the icons, it was unclear to them what the different icons signified. The fact that the error messages were available by hovering over the icons was not immediately clear to the users and one user tried to click the icon. On the other hand, the users had no problems with the validation error messages shown at the top of the traditional web form after submission. This is probably because this seems to be the norm for displaying error messages, but it would also indicate that error messages should be displayed clearly in order for the user to notice and react to them. This feedback has resulted in changes in the web implementation to display the error messages directly beneath the field the error concerns, as can be seen in Figure 1(a). The green icons indicating correct fields have also been removed to avoid too much visual overload.

Another issue identified during the user testing was how to indicate the expected format. The tested version did not indicate the expected format, which especially caused problems with date fields, as dates can be formulated in a number of different ways. This observation resulted in showing the expected format next to the date field in the current version.

An important observation during the user testing was how the form was filled out top to bottom, under the assumption that the fields already filled out would not change. This is as expected, but is important to remember when defining

forms and avoid creating rules that will change fields already filled out “on the way down”. The lessons from this evaluation are mainly about the construction of the UI, which is not directly the focus of the framework. The evaluation does however show the usefulness of the idea behind the framework and the use of adaptive forms. A majority of users in the evaluation liked the adaptive forms after having adjusted themselves to the different mindset of getting instant feedback and not having to wait for submission validation.

Developer Feedback. As a way of getting some feedback from developers who were to use the framework in their development work, the framework was given to a developer to evaluate. The developer was given the framework including the UI web implementation and was asked to develop a web form. The developer found the learning curve of using the web implementation quite steep due to the many Ajax HTTP calls exchanged between the client and the server. It needed a different way of thinking about control in the servlet environment and a change to letting the hooks do much of the work. This would indicate that the control flow should be made more clear in the future along with some more detailed tutorials. The work with defining the forms using XML worked well and the structure was quickly grasped. The main complaint was the lack of a more convenient way of verifying the correctness of the form, prior to instantiation. This could be solved by including a tool in the framework that would control the form against the XML Schema available and at the same time ensuring that referenced JavaBeans actually exist. As for the use of the validation rules, the simple rules were easily understood and after looking into XPath, which was unknown to the developer beforehand, this was also easily understood. One complaint was however how this would fit into already defined business logic without having to write the same thing twice. One way of avoiding defining business rules more than once could be to rely mainly on JavaBeans and then reuse these beans throughout. The general response to the use of the framework, was that after passing the initial hurdle of learning the work flow of the framework, it can save a developer a lot of time, compared to having to writing the validation of client and server side by hand. One of the authors (not the original developer of AdapForms) have utilised the AdapForms framework in the context of the eGov+ project, as reported in [2,3]. Our experience with the framework closely resembles those reported above.

Form Handler Feedback. The framework has also been used in connection with the parental leave case of the eGov+ project and included in the prototype developed in connection with this case and described in [2,3].

In connection with the development of this prototype, the concept of adaptive documents was introduced to the caseworkers set to handle the incoming forms. They were generally positive about adaptive documents and some effort was made trying to annotate existing physical documents, in order to establish relationships between fields.

The caseworkers found it challenging to analyse the existing documents and identify fields which were e.g., dependent on former input or pairs of fields depended on each other. This shows that the change from creating one static form to handle all potential users and their input and then designing a form which is capable of adapting based on input such as gender or user role is non-trivial. This challenge is something that has to be worked out between the developers in charge of developing the form and the people with the domain knowledge specific to the form in question.

5 Related Work

Form validation is a intrinsic part of form processing, and has been researched both on the desktop and in web-based settings. FormGen [5] is a Java GUI tool, which can generate a dynamic form based on data structured as a context-free grammar (CFG). FormGen uses this CFG to generate Java classes for both data model and GUI components. As in AdapForms, the data model in FormGen is a tree, and as the user adds data to the form, new nodes are added to the tree.

Dynamic Forms [7] is an example of a validating form generator using the Form Descriptive Language (FDL). FDL can be edited with an interactive editor written in FDL. When users are interacting with the finished form, Dynamic Forms provides feedback of completion and validity to the user by colour coding. The idea of colour coding is also known from other frameworks and is also used in the example implementation of the web user interface for AdapForms.

XForms [9] is a W3C alternative to the common HTML form. XForms is, as AdapForms, XML based, and differs from regular HTML forms by separating data and markup. Thus, one form can be specified and used several different places with different markup. XForms extends the standard HTML form with validation and adaptation. The validation can be performed using XML Schema. Types are defined using XML Schema, and input can later be validated against these types. XForms can also mark input as read-only, required, or relevant. Adaptation of the document can be accomplished by changing the value of a field's relevance at run-time. The GUI may hide all irrelevant fields as not to confuse the user. AdapForms have the same capabilities for in-document validation and adaptation, but extends these capabilities by allowing more complex validation and adaptation within the framework, such as using database look-ups and other server-side techniques, as described in Section 2.2 and 2.3.

JavaScript is widely used to make regular static HTML forms more responsive and adaptive. Such JavaScript can however be tedious and error prone to develop, which is addressed by PowerForms [4] by supporting client side validation for user input in HTML forms. It generates an interactive form based on a HTML static form and an PowerForms specification. The validation is continuous, showing the validity as the form is filled. The status is again indicated by red, yellow or green. The rules are constructed using a simple **if-then-else** structure, or using regular expressions. Regular expressions are used to validate the format of the text being entered. The rules are transformed into a finite-state machine

expressed in JavaScript, which validates the form input. The **if-then-else** constructs can be chained to make the validity of fields depend on other fields.

6 Conclusion

Forms are an integral part of modern living, yet the basic form has changed little over the decades—it may these days reside on a web page, but the general form remains the same. Validation of forms is usually either a lengthy process (fill-out, submit, review error messages, revise, resubmit), or relatively simple-minded using specialised JavaScript code. We have in this paper presented a general framework to define forms, designate acceptable field values, the inter-relationships between fields, as well as an architecture to transparently validate, report state of, and adapt a form as the user is filling it out. Depending on the needs of the developer, the validation can be simple (type and range checks), XPath based, or advanced such as integrating server-side business logic through JavaBeans. Depending on the user's input, the form can adapt itself, freeing the user from having to deal with an over-general form covering all possible permutations. Our initial evaluation leaves us hopeful that this is a valid approach, though there certainly still are many aspects of effectively communicating expected values and form state to the user that need to be explored. Likewise, the prototype implementation, while suited for our experiments has room for improvement, notably in terms of scalability.

Acknowledgments

The eGov+ project is financed by a grant from NABIIT, the Danish strategic research programme for nano-, bio-, and IT-sciences.

References

1. Berglund, A., Boag, S., Chamberlin, D., Fernández, M.F., Kay, M., Robie, J., Siméon, J.: XML path language (XPath) 2.0, W3C recommendation. Tech. rep., The World Wide Web Consortium, W3C (2007)
2. Bohøj, M., Borchorst, N.G., Bouvin, N.O., Bødker, S., Zander, P.O.: Time collaboration. In: Proceedings of the 28th International Conference on Human Factors in Computing Systems. ACM, Atlanta (2010)
3. Bohøj, M., Bouvin, N.O.: Collaborative time-based case work. In: Proceedings of the Hypertext Conference 2009, pp. 141–146. ACM, New York (2009)
4. Brabrand, C., Møller, A., Ricky, M., Schwartzbach, M.: Powerforms: Declarative client-side form field validation. World Wide Web 3(4), 205–214 (2000)
5. Brandl, A., Klein, G.: FormGen: A Generator for Adaptive Forms Based on EasyGUI. In: Proceedings of HCI International Human-Computer Interaction: Ergonomics and User Interfaces, vol. 99, pp. 22–26 (1999)

6. Gammelmark, H.: Adaptive Forms. Master's thesis, Department of Computer Science, Århus, Denmark (April 2009),
<http://adapforms.gammelmark.eu/files/adapforms-thesis.pdf>
7. Girgensohn, A., Zimmermann, B., Lee, A., Burns, B., Atwood, M.: Dynamic forms: An enhanced interaction abstraction based on forms. In: Proceedings of Interact 1995, pp. 362–367. Chapman & Hall, Boca Raton (1995)
8. Ørbaek, P.: Programming with hierarchical maps. Tech. rep., Aarhus University, PB-575 (2005), <http://www.daimi.au.dk/publications/PB/575/PB-575.pdf>
9. W3C: XForms 1.1 (2007), <http://www.w3.org/TR/xforms11/>

Design and Implementation of Linked Data Applications Using SHDM and Synth

Mauricio Henrique de Souza Bomfim and Daniel Schwabe

Informatics Department, PUC-Rio Rua Marques de Sao Vicente, 225.
Rio de Janeiro, RJ 22453-900, Brazil

mauriciobomfim@gmail.com, dschwabe@inf.puc-rio.br

Abstract. In this paper, show how Linked Data Applications (LDAs) can be designed and implemented using an evolution of the Semantic Hypermedia Design Method, SHDM, and a new development environment supporting it, Synth. Using them, it is possible to take any RDF data available on the Linked Data cloud, extend it with one's own data, and provide a Web application that exposes and manipulates this data to perform a given set of tasks, including not only navigation, but also general business logic. In most cases, the only code that needs to be written is for the Business Logic; the remainder code is automatically generated by Synth based on the SHDM models.

Keywords: Linked Data, Semantic Web, Linked Data Applications, RDF, Design Method, Model Driven Development.

1 Introduction

Up until recently, Web applications followed, to a great extent, the traditional data intensive development approaches. The prevailing paradigm is one where organizations or individuals are responsible for either creating or collecting, through the application itself, all its data, which remains under its management. With the advent of the Semantic Web, and especially the more recent growth of the Linked Data initiative [2], we are witnessing the emergence of the Linked Open Data (LOD)¹ cloud, a collection of interlinked data sources spanning a wide range of subjects. It is now quite feasible to design Web applications (hosted in websites) whose contents are at least partially drawn from the LOD cloud. For example, several sites at the BBC (e.g. 2010 World Cup website²) routinely make use of data pulled from the LOD cloud.

This new scenario has given rise to a new challenge – how to effectively build applications that consume linked data, often combining with locally generated and managed data – that we will call “Linked Data Applications”, LDAs in short. At first sight, this may sound similar to the problem of building traditional Web applications – after all, the Semantic Web is part of the traditional, so-called “Web of documents”. For such Web applications a number of design methods have been proposed, e.g., OOHDM [9], SHDM [4], WebML [2], UWE [3], Hera [13], among others.

But LDAs present additional challenges. First, one of its basic tenets is the reuse of existing information – both vocabularies and instance data, whenever possible.

¹ <http://linkeddata.org>

² http://www.bbc.co.uk/blogs/bbcinternet/2010/07/bbc_world_cup_2010_dynamic_sem.html

Second is the principle that the data should carry as much machine-processable semantics as possible. Following this principle, it should be expected that, within feasible limits, the application semantics also be captured in machine processable way.

From the model-driven design (MDD) [11] point of view, this is not a new idea. Software development, according to MDD, is a process whereby a high-level conceptual model is successively translated into increasingly more detailed models, in such a way that eventually one of the models can be directly executed by some platform. To be consistent with the LDA philosophy then, they should be specified using models expressed in the same formalisms used to describe the data itself.

There are some proposals of development environments or frameworks for supporting the development of LDAs, such as CubicWeb³, the LOD2 Stack⁴, and the Open Semantic Framework⁵. In addition, semantic wiki-based environment such as Ontowiki⁶, Kiwi⁷, and Semantic Media Wiki⁸ have also been used as platforms for application development over Linked Data.

While useful, they do not present a set of integrated models that allow the specification of an LDA, and the synthesis of its running code from these models. Therefore, much of the application semantics, in its various aspects, remains represented only in the running implementation code.

On the other hand, existing development methodologies, such as OOHDM, WebML, UWE, and Hera, do not have the primitives to directly implement LDAs, although some are able to import and manipulate RDF data.

In this paper we present an updated version of SHDM [7], the Semantic Hypermedia Design Method, and Synth⁹, a development environment that allows the specification of LDAs according to SHDM models, and the generation of running code from this specification. Using Synth, it is possible to generate LDAs with little or no programming, simply by declaring models as proposed by SHDM.

The remainder of this paper is organized as follows. Section 1 presents a motivating example that will be used throughout the paper to illustrate the various issues being discussed; this application is built over data published in the Semantic Web, enriching it with one's own private data. Section 2 presents a brief summary of SHDM, focusing on its proposed models and discussing their relevance for LDAs. Section 3 presents Synth, the development environment supporting SHDM, and shows how the example application has been implemented. Finally, Section 4 discusses related and future work.

2 A Working Example

In order to facilitate the presentation and discussion of our approach, we first present an example LDA that illustrates a typical scenario. We will use it as a

³ <http://www.cubicweb.org>

⁴ <http://lod2.eu/WikiArticle/TechnologyStack.html>

⁵ <http://openstructs.org/open-semantic-framework>

⁶ <http://ontowiki.net/Projects/OntoWiki>

⁷ <http://www.kiwi-project.eu>

⁸ http://www.semantic-mediawiki.org/wiki/Semantic_MediaWiki

⁹ <http://www.tecweb.inf.puc-rio.br/synth>

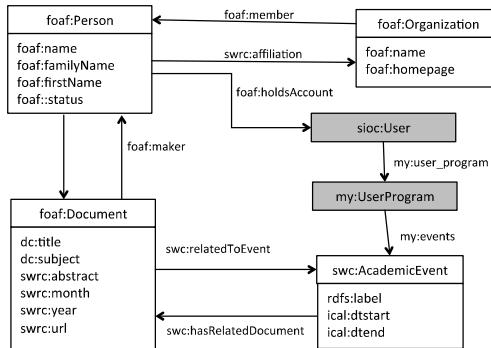


Fig. 1. Domain model for a Personalized Conference Schedule application

reference to exemplify certain aspects of the proposed approach in the remainder of the paper.

Many conferences make the metadata about its events (keynote talks, technical and scientific paper presentations, tutorials, etc.) publicly available as RDF data, such as the Semantic Web Conference series (see data.semanticweb.org). As is often the case, a conference attendee must choose which events she is interested in, and assemble a personal schedule of events to attend. We show a simple application that allows the attendee to use the available conference metadata published in the LOD to build a personal schedule.

Event Details
Opening Ceremonies When: 11/09/2010 08:30 to 09:00 Where: Auditorium
Justification Oriented Proofs in OWL When: 11/09/2010 10:30 to 11:00 Where: Room 3B
Semantic Recognition of Ontology Refactoring When: 11/09/2010 10:30 to 11:00 Where: Auditorium
Using SPARQL to Test for Lattices: application to quality assurance in biomedical ontologies When: 11/09/2010 10:30 to 11:00 Where: Yellow River
Doctoral Consortium/I: Mining and Search When: 11/09/2010 10:30 to 12:00 Where: Room 3C/D
In-Use Track/I When: 11/09/2010 10:30 to 12:30 Where: Yellow River
Research Track: Ontology Evolution When: 11/09/2010 10:30 to 12:30 Where: Auditorium

Fig. 2. A list of events at a conference, with the option to select for inclusion in personal schedule

Figure 1 shows the Domain model of the example application, using an UML-like notation. Classes in white occur in the input data taken from data.semanticweb.org (only a small fragment of the entire vocabulary was needed, depicted in this schema). The classes in gray were added to model the semantics of the example application.

The *sioc:User* class models the user of the application, and the *my:UserProgram* class models the personalized program, which will be a set of *swc:AcademicEvents*.

The screenshot shows a web interface for the 'Synth Semantic Web application development environment'. At the top, there's a navigation bar with tabs for 'Events', 'Persons', 'Publications', and 'My Program' (which is currently selected). Below the navigation bar, there's a search bar and a 'Check the events of your interest' section. This section lists several academic events with checkboxes next to their names. Some events have their details (When, Where) displayed below them.

Event Name	When	Where
<input checked="" type="checkbox"/> Opening Ceremonies	11/09/2010 08:30 to 09:00	Auditorium
<input type="checkbox"/> Invited Talk: me schrafel	11/09/2010 09:00 to 10:00	Auditorium
<input checked="" type="checkbox"/> Justification Oriented Proofs in OWL	11/09/2010 10:30 to 11:00	Room 3B
<input checked="" type="checkbox"/> Semantic Recognition of Ontology Refactoring	11/09/2010 10:30 to 11:00	Auditorium
<input checked="" type="checkbox"/> Using SPARQL to Test for Lattices: application to quality assurance in biomedical ontologies	11/09/2010 10:30 to 11:00	Yellow River
<input type="checkbox"/> Exploiting Relation Extraction for Ontology Alignment	11/09/2010 10:30 to 11:00	Room 3C0
<input checked="" type="checkbox"/> Doctoral Consortium/I: Mining and Search	11/09/2010 10:30 to 12:30	Room 3C0
<input checked="" type="checkbox"/> In-Use Track/I	11/09/2010 10:30 to 12:30	Yellow River

Fig. 3. A personal schedule of events

The starting point in the application is an interface that lists all events in chronological order, as illustrated in Figure 2. For each event, the user has the option of checking a checkbox to indicate her interest.

Once the user has selected the events of interest, a personal program is shown in Figure 3

The screenshot shows a detailed view of an event from the previous list. The top navigation bar and search bar are visible. On the left, there's a sidebar with a 'Events by User Program' section and a 'Filter' input field. The main content area displays the details of the 'Justification Oriented Proofs in OWL' event. It includes sections for 'rdfs:label', 'When', 'Where', 'Presenters', 'Name', and 'Organizations'.

Label	Value
rdfs:label	Justification Oriented Proofs in OWL
When	11/09/2010 10:30 to 11:00
Where	Room 3B
Presenters	Matthew Horridge Ulrike Sattler Björn Parviz
Name	
Organizations	University of Manchester

Fig. 4. Details of an event

In this interface, the user can click on an event name to see details, as illustrated in Figure 4. In this interface, clicking on a person’s name will lead to an interface showing more data about that person. Clicking on an organization’s name will show all events associated to that organization.

Notice that there is no direct relation in the Domain Model between *Organization* and *AcademicEvent*; rather, it must be computed as the *Organization* to which the *Authors* of the *Document* presented in the *AcademicEvent* belong.

We next discuss new or updated features of SHDM that deal with modeling LDAs.

3 The Evolution of SHDM

SHDM is a model-driven approach to design Semantic Web applications. Since it was first formulated, the Semantic Web itself has evolved, e.g., with the advent of the LOD cloud. Consequently, we have updated several aspects of SHDM, discussed in this section.

SHDM includes six different phases: Requirements Gathering, Domain modeling (formerly Conceptual Modeling), Hypertextual Navigational modeling (formerly Navigation Modeling), Abstract Interface modeling; Business Logic modeling (which did not exist previously), and Implementation. Each phase focuses on a particular aspect and produces artifacts detailing the application to be run on the web.

Before looking at SHDM details, we first discuss what MDD means in the context of Linked Data.

3.1 Linked Data, Applications and MDD

Linked Data, by itself, focuses only on providing information encoded as RDF, possibly following an RDFS or OWL schema, in such a way that it is possible for a program to process each information item, and to obtain additional related information when available. Thus, it provides but a small portion of the necessary specifications needed build a full-fledged application.

The only semantics of an RDF graph, is given by the RDF (and RDFS) meta-model semantics as specified in the standard¹⁰. This captures a very limited fraction of the “broad meaning” of the statements in the graph. The semantics restrict the possible interpretations of the set of triples as referring to some “real world” entity or objects, and their relations. The programs that manipulate these statements add additional specific semantics, as their behavior is determined in some way by particular sets of triples they receive as inputs.

In MDD, a set of models determines the ultimate application behavior. Each model captures some aspect of the application – e.g., data (domain), interface, hypertextual navigation, etc. If these models are specified as RDF statements using some vocabularies, specific interpreters that implement the desired behavior must provide the intended semantics. In fact, for RDFS and OWL, inference engines play this role with respect to the data semantics, but there is no counterpart for application behavior (i.e., the business logic).

¹⁰ <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>

In SHDM, each of the proposed models establishes a specific vocabulary, and the semantics are currently given through the code generated using the Synth environment, presented later in this paper.

We will next describe the more relevant models in SHDM, as they apply to LDAs, pointing out evolutions with respect to the originally proposed method.

3.2 Domain Modeling

The Domain model characterizes the universe of discourse of a particular application. In most proposed methodologies, the (equivalent to the) Domain model is specified through a Class Schema (in OO methods) or ER schema (in database oriented methods). In contrast, in SHDM a Domain model is simply a set of RDF triples, which form a graph, which may include RDFS or OWL definitions. In the extreme case, there may not exist any Class definitions in Domain Model, just instances of resources representing information items.

This is an evolution over earlier versions, where the domain model was specified in SHDM's own vocabulary. Therefore, SHDM can use any Conceptual Modeling technique that is capable of producing RDF graphs, or none at all, if the starting point is an already existing RDF graph.

However, since we are focusing on LDAs, it is necessary to add an additional abstraction to SHDM's metamodel, to capture the concept of “datasets”. Datasets are described using the VOID vocabulary¹¹, which was designed precisely for this purpose. Using datasets, it is possible to use RDF graphs stored in the LOD cloud as part of an application's Domain Model. We will later show how this is achieved.

3.3 Business Logic Modeling

LDAs – indeed, all applications - are built to perform computations over data instances of the Domain Model, causing effects that change the data and present some results to the user. This computation is usually defined to support a set of tasks and users, the intended audience of the application, which sometimes can be quite broad. The Business Logic of the application specifies these computations.

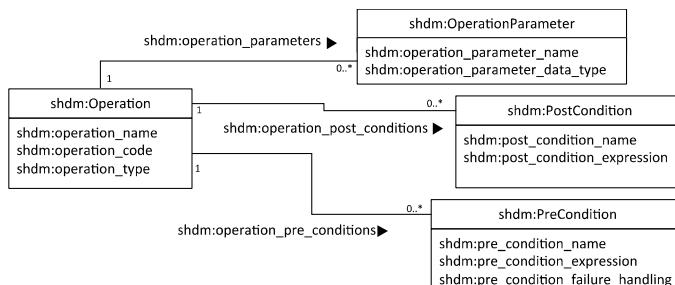


Fig. 5. The Operation metamodel in SHDM

¹¹ <http://vocab.deri.ie/void/guide>

In SHDM, the Business Logic model is given by specifying a set of *Operations* that will be made available by the application, much in the same way as services in service-oriented applications. SHDM's metamodel for Operations is shown in Figure 5. It is similar to Web Services specifications in other vocabularies (e.g., OWL-S¹², WSM¹³), with a few additions, described next.

Operations have properties specifying *Parameters*, and *Pre-* and *Post-conditions*. The *shdm:operation_name* property provides a label for the operation. The *shdm:operation_type* can be either “Internal” or “External”. The latter are operations that can be invoked from outside the application, i.e., they are visible on the Web and may be invoked as a REST service. The former are operations that can only be accessed from within the application and are not visible outside..

The actual behavior specification of the operation is given in the *shdm:operation_code* property. The SHDM metamodel is agnostic as to how this code is given, so the value of this property is currently specified as a string. In principle, this string can be code in any language for which there is an interpreter. This can be a general programming language, or a Domain Specific Language [12], such as BPEL¹⁴ or BPMN¹⁵. Strictly speaking, the value of this property could also be another RDF graph representing a particular DSL (e.g. BPMN), but we have not explored this. It should also be noted that Operations may be called from within other Operations.

The effective implementation behavior of the application is achieved by integrating the code DSL interpreter with the runtime engine – i.e., it must be able to access and change the values of the data.

3.4 Hypertextual Navigation Modeling

The term “Navigation Design” or “Hypertext Design” has been used to designate the activity of designing the “navigation topology” of Web applications [9], which also applies to LDAs. The main idea is that applications aim at supporting a given set of tasks and, while performing a task, some of its steps are often best supported by providing a hypertextual navigation structure over the information items being processed.

For some domains, such as online newspapers and magazines, browsing and navigating is the main task to be performed, and hypertextual navigation is well suited to support it. For other domains, e.g. online stores, hypertextual navigation is useful to support only a few of the tasks, e.g., browsing the catalog section, but is of little help to support other tasks, e.g., the transactional (e.g., payment) processing.

Hypertextual navigation modeling prescribes preferred navigation paths through the information items to support a given set of tasks. When specifying the navigation options, it is useful to be able to refer to sets of items that share similar navigation alternatives, instead of individual ones. For example, it is better to state that “when accessing any *Person*, one may navigate to one of the *Papers* s/he has created, and from any *Paper*, continue navigating to other *Papers* created by that same *Person*”, instead of specifying this linking structure repeatedly for every instance of *Person* and *Paper*. Notice that several hypertextual links are induced by the task, and are not present in the Domain Model – e.g., “next” and “previous” *Paper* created by a *Person*.

¹² <http://www.w3.org/Submission/OWL-S/>

¹³ <http://www.wsmo.org>

¹⁴ <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>

¹⁵ <http://www.bpmn.org>

Following OOHDM, the major primitive for specifying the hypertextual navigation topology of LDAs is the *Context*. A *Context* is a set of resources that share similar hypertextual alternatives, e.g. “Papers by a Person”, “Products in a Department”, and “Stories in a Section”. In a way, a Context plays the analogous role with respect to hypertextual navigation as Classes (in OO languages) play for structure and behavior. The same way the Class definition determines the structure and behavior of its instances, the Context definition determines the navigation alternatives of its elements.

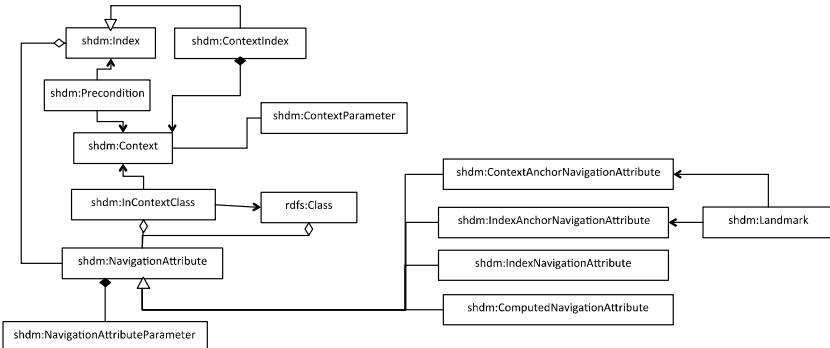


Fig. 6. SHDM Hypertextual Navigation metamodel

RDF data constitutes a graph, which can be considered as a “node-and-link” hypertext. For LDAs, this “node-and-link” model is often too low level, not supporting the set-based specification of contexts. In other words, it is necessary to extend the pure RDF graph of the Domain Model with additional properties that will drive the higher-level Hypertext navigation.

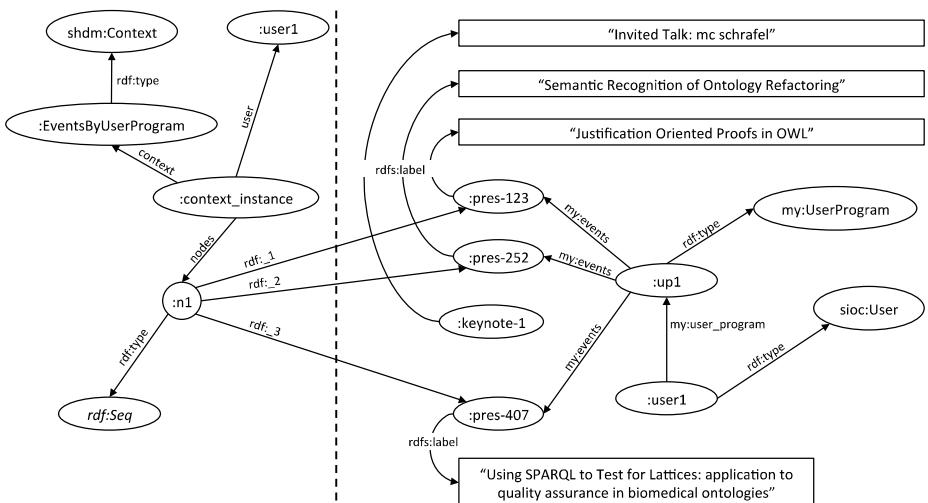


Fig. 7. A fragment of the hypertextual navigation graph (left side) for the example

Accordingly, SHDM's navigation model is defined as a set of additional properties assigned to RDF resources in the Domain model. The hypertextual navigation behavior defined by this model is implemented as part of the runtime environment of Synth. Figure 6 shows the Hypertextual Navigation metamodel for SHDM.

A full description of SHDM's Hypertextual Navigation metamodel is outside the scope of this paper; more details can be found in [4]. Nevertheless, we illustrate how the Domain model is extended with navigation properties at runtime, to allow the interpreter to implement the hypertextual navigation behavior specified in the corresponding model.

Figure 7 shows a fragment of the data graph of our example application on the right hand side, and the hypertextual navigation runtime structure on the left hand side. This fragment shows the navigation structure of the instance of the “Events-ByUserProgram” context (a “context_instance” node) for “user1”, which is the value of the “user” parameter of the context. In this case, “pres-12”, “pres-252” and “pres-407” are part of this context. Notice that the “context_instance” node is used to group the context elements and to provide ordering among them.

A more careful analysis of what actually *is* hypertextual navigation reveals that it is ultimately just one (more) kind of application behavior, among all behaviors defined by the Business Logic. However, differently from any arbitrary behavior one might have in an application, which usually is domain-dependent, the hypertextual navigation behavior has a well-known semantics, which should also be modeled as an operation in the Business Logic model. In other words, what navigation means is known *independently* of any particular domain, in advance.

From this observation, we define the hypertextual navigation in the development support environment simply as a *pre-defined* set of operations. In other words, navigation operations are operations whose code is predefined, and automatically integrated into the generated runtime environment.

There are other advantages in to modeling navigation as operations. For example, since operations have pre-conditions, it is possible to define conditional navigation behavior. For instance, in our example, if the user has not yet selected her preferred events, if she tries to navigate to the “My Program” index, the application will present a message and request the user to first select the preferred events.

3.5 Interface Modeling

The Interface Model in SHDM [15] is based on the idea that it is possible to separate the “essence” of the interface from its look-and-feel. This is achieved by decomposing the interface specification into an Abstract Interface Model, and a Concrete Interface Model.

In brief, the Abstract Interface focuses on the roles played by each interface widget in the information exchange between the application and the outside world, including the user. It is abstract in the sense that it does not capture the look and feel, or any information dependent on the runtime environment. The Concrete Interface model is responsible for the latter.

Summarizing the Abstract Interface meta-model, an (abstract) interface is a composition of abstract interface elements (widgets). These in turn can be an *ElementExhibitor*, which is able to show values; and *IndefiniteVariable*, which is able to capture

an arbitrary input string; a *DefinedVariable*, which is able to capture input values (one or several) from a known set of alternatives; and a *SimpleActivator*, which is able to react to an external event and signal it to the application.

From an *Abstract Interface*, a mapping specification made by the designer determines how each abstract widget will be mapped onto one or more *Concrete Interface* elements, and onto which *Operations*. Notice that the mapping to *Operations* unifies access to the Domain Model - if it is a CRUD operation of an element of the Domain; to the Navigation Model, if it is a Navigation operation; and to the general Business Logic, otherwise.

This model allows a cleaner separation between the Hypertext Navigation and Business models; there is no need to include both types of primitives in a single graph, as proposed in WebML or UWE. When an event (i.e., user click) is captured at the interface (via a *SimpleActivator* widget), the mapping to the operations (there can be more than one operation mapped to the same interface element) will determine which operations will be executed, regardless of their being navigation or not.

We have subsequently extended this model to allow specifying interface-only behavior, thus modeling modern Rich Application Interfaces [5], but we will not detail here for reasons of space.

We next present Synth, a development environment that supports the creation of LDAs using the SHDM method.

4 The Synth Development Environment

Synth is a development environment for building applications that are modeled according to SHDM. It provides a set of modules that receives, as input, models generated in each step of SHDM and produces, as output, the hypermedia application described by these models. Synth also provides an authoring environment that facilitates the adding and editing of these models through a GUI that can run on any web browser.

4.1 Software Architecture

The software architecture of Synth was designed to be independent of its implementation technology. It consists of a set of modules, each responsible for maintaining and interpreting one of the models generated in each phase of SHDM. Each module is composed by a model described in a corresponding ontology in RDFS or OWL, and an interpreter that gives semantics to the models, in addition to the basic semantics of RDFS and OWL, in which they are represented. These modules work together, interpreting their models and communicating with each other, in order to generate the application runtime in accordance with the definitions of each model.

Figure 8 shows a conceptual view of the modular software architecture for applications modeled using SHDM. The gray boxes represent the modules and the white boxes represent the components of each module.

The persistence module handles the access and manipulation of application data. This module is composed of two layers: a storage, inferences and query layer and a RDF(S)/OWL mapping layer.

The storage, inferences and query layer provides a single interface for accessing multiple environments and platforms for RDF data. This layer converts the access interfaces of various RDF data environments (e.g., Jena¹⁶, Sesame¹⁷, Virtuoso¹⁸ or OWLIM¹⁹) into a single interface known by the RDF(S)/OWL mapping layer, by applying the Adapter design pattern. This layer is also responsible for distributing queries among various data repositories, local or remote, combining their results, as enabled under the Linking Open Data initiative.

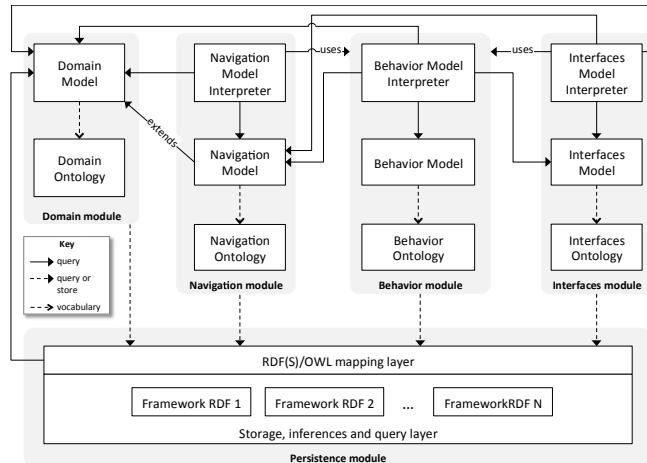


Fig. 8. Conceptual view of the Synth software architecture

The RDF(S)/OWL mapping layer provides a view of the data and meta data, originally persisted as RDF triples, as primitives of the programming language in which the application is implemented.

The other modules are the domain, navigation, business logic, and interfaces modules. Each of these maintains and interprets its corresponding models, and is similarly structured. The only exception is the domain module, which has no interpreter because the semantic of the domain model is that of RDFS and OWL, and the persistence module provides an interpreter for them in the RDF(S)/OWL mapping layer.

4.2 Module Collaboration

To illustrate how the modules collaborate, Figure 9 presents the sequence diagram showing the events in a typical execution of a “navigate” hypertextual navigation operation. The user interaction always happens through the external operations of the business logic model, available as Web Services. External agents invoke external operations by sending HTTP requests to the application. In this sense, external

¹⁶ <http://jena.sourceforge.net>

¹⁷ <http://www.openrdf.org>

¹⁸ <http://virtuoso.openlinksw.com/>

¹⁹ <http://www.ontotext.com/owlim/index.html>

operations fulfill the same role as the “controller” in the MVC-based (model-view-controller) architectures, coordinating user interactions, obtaining data from the domain Model, instructing the View to generate the interface and, finally, delivering the result to the user or external agent.

In this sequence diagram, the user sends a message to the business logic module invoking the method “execute” e informing the name of an operation, in this case “context”, and some parameters for this operation. This operation performs the “contextual navigation” operation of the SHDM hypertextual navigation model.

After that, the external operation invokes the method “get_context” from the navigation module with the context identifier “AllPerson” as a parameter. The navigation module retrieves the “AllPerson” context definition, obtaining the query expression. This expression is then used to invoke the method “dsl” from the domain module, which simply passes it to the persistence module. This module converts this expression to an equivalent Federated SPARQL query expression, executes it getting the results as RDF triples, and maps these results into the programming language primitives.

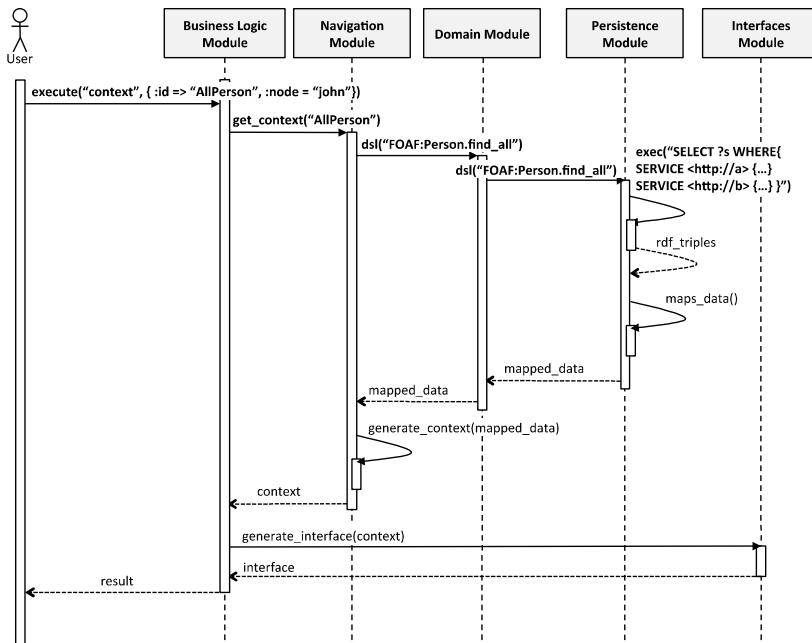


Fig. 9. Collaboration between modules in Synth

This mapped data is returned to the domain module and passed to the navigation module, which generates the context instance with its nodes based on the received data, and returns this context instance to the business logic module. Then, the business logic module invokes the method “generate_interface” in the interfaces modules passing the context instance as parameter, which returns the actual concrete interface. Finally, the business logic module returns this result to the user.

4.3 Implementation Architecture

Synth is implemented with Ruby on Rails, an MVC framework for web applications development. Within the MVC architecture, it maintains a modular organization where each module presented in the conceptual view shown in Figure 8 is implemented as a composition of one or more components of the MVC general implementation architecture.

All data in Synth is maintained as an RDF graph. It includes not only the instances of domain data, but also the metadata about the models and meta-models of SHDM, expressed as RDF(S) and OWL. This data is manipulated programmatically using the Jena framework as an API. It plays the role of the storage, inference and query layer in the persistence module in conceptual architecture. ActiveRDF [7], is a library for accessing RDF from Ruby programs, mapping the RDF data into Ruby primitives, playing the role of the RDF(S)/OWL mapping layer. ActiveRDF provides an API that facilitates CRUD operations within Ruby programs, and has an adapter for Jena.

Our previous experience (see [6]), and Oren et al [7], explain the reasons we have chosen Ruby and Rails as the implementation environment.

4.4 Authoring Environment GUI

Synth provides an authoring environment GUI using HTML forms that can be accessed from a web browser and allows the creation and editing of primitives of SHDM models. It is possible run the application while it is being built using this interface, validating it in each step of the development process.

Synth also provides the RDF Scaffold, a generic RDF browser and editor that allows the execution of CRUD operations over the local application RDF database, in the same way that it can be done in some well known RDF browsers and wikis like Tabulator, Disco²⁰ or OntoWiki.

4.5 DSLs within Synth

Synth provides several ways to specify the selection rules that determine the set of resources that compose a context; these rules are expressed as context query expressions. These query expressions can be specified in three alternative query languages: directly in SPARQL; in a DSL hosted in Ruby which is defined by the ActiveRDF framework; or in SynthQL, a customized simplified query language created specifically for Synth, which can represent the most common context queries in typical LDAs.

The accepted SPARQL syntax is the same accepted by the ARQ query machine, part of Jena framework, which supports the Federated SPARQL query that is important for specifying datasets in the context queries. The ActiveRDF DSL is similar to the one described in [6]

The goal of SynthQL context query language is to abstract the syntax of SPARQL for the most commons query expressions typically found in LDAs. This approach tries to minimize the need for knowledge about SPARQL, while still covering a large set of common expressions with a simple syntax. The particular DSL is similar to the one presented the previous version of Synth, HyperDE [6].

²⁰ <http://www4.wiwiss.fu-berlin.de/bizer/ng4j/disco/>

The code below is a context query expression in SynthQL. This query means “All resources whose rdf:type is foaf:Document, dc:title contains a substring “owl”, foaf:maker has the same value as the parameter ‘person’ passed during the user navigation, ordered by dc:title, limited to 100 results, and the query should be evaluated on the datasets identified by “local” and “iswc2010”.

```
selects {
  type FOAF::Document
  dc::title like "owl"
  foaf::maker person
  order DC::title
  limit 100
  datasets :local, :iswc2010}
```

4.6 Example Application

This section shows how the example LDA described in Section 1 is implemented in Synth. For reasons of space, we will not detail the Interface Model for this application; suffices to say that the example interfaces shown there use the default interface generated by Synth for any application.

The raw RDF data from <http://data.semanticweb.org/conference/iswc/2010/complete> was imported to the Synth local database. This data is the core Domain Model of the application, which is enriched with the customized schedule.

After the data importing, the navigation specification was entered into Synth through its GUI (see Figure 10). The code below shows the specification of the context swc:AcademicEvent “byUserProgram”.

```
:byUserProgram a shdm:Context ;
  shdm:context_name "byUserProgram";
  shdm:context_title "Events by User Program";
  shdm:context_query "user.my::user_program.my::events";
  shdm:context_parameters [ a SHDM::ContextParameter;
    shdm:context_parameter_name 'user'];
```

This specification uses the DSL hosted in Ruby defined by the ActiveRDF API. The Figure 10 shows the Synth interface to edit this specification. The resulting context navigation screen is shown Figure 4.

The list below is part of the specification of the “EventsByUserProgram” index. In this example some navigational attributes are omitted to save space, but they are very similar to the navigational attributes shown above.

```
:EventsByUserProgram a shdm:ContextIndex ;
  shdm:index_name "EventsByUserProgram";
  shdm:index_title "My Program";
  shdm:context_index_context :byUserProgram;
  shdm:context_anchor_attributes [
    a shdm:ContextAnchorNavigationAttribute;
    shdm:navigation_attribute_name "label";
    shdm:navigation_attribute_index_position 1;
    shdm:context_anchor_label_expression "self.rdfs::label";
    shdm:context_anchor_target_context :byUserProgram;
    shdm:context_anchor_target_node_expression "self"];
```

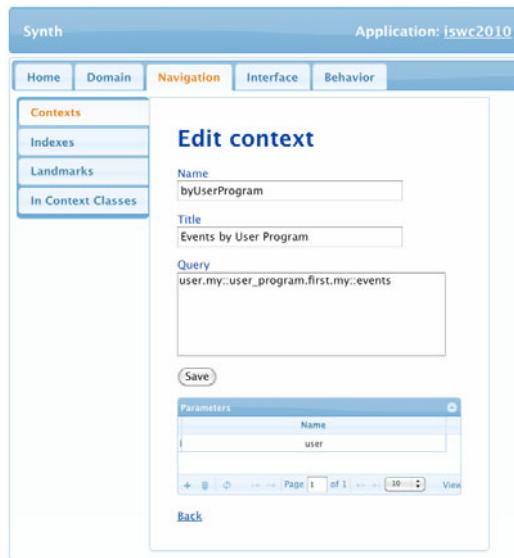


Fig. 10. Synth interface for “byUserProgram” context specification

This index is based on the context “byUserProgram” and has the navigational attribute that is a context anchor. This context anchor shows the rdfs:label of the swc:AcademicEvent and the target is the resource in which the current index entry is based on the context “byUserProgram”.

5 Conclusion and Future Work

We have presented the evolution of SHDM, and Synth, a new development environment that supports the design and implementation of LDA, mixing external and locally created RDF data. The major changes in SHDM are: Domain Model defined by any RDF graph, including graphs distributed over several repositories; a Business Logic Model that unifies Hypertextual Navigation and other application functionalities; and a Hypertextual Navigation Model as an extension of any RDF graph.

Future work will add new models to SHDM, and their corresponding integration in Synth, such as authorization, transactions, and adaptation.

Acknowledgement. The authors were partially supported by grants from CNPq and Petrobras.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data – The Story so Far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
2. Ceri, S., et al.: Designing Data-Intensive Web Applications. Morgan Kaufmann, San Francisco (2003)

3. Koch, N., Kraus, A.: The Expressive Power of UML-based Web Engineering. In: 2nd Int. Workshop on Web-Oriented Software Technology (IWWOST 2002), CYTED, Málaga, Spain, pp. 105–119 (2002)
4. Lima, F., Schwabe, D.: Application Modeling for the Semantic Web. In: Proceedings of LA-Web 2003, Santiago, Chile, pp. 93–102. IEEE Press, Los Alamitos (2003)
5. Luna, A., Schwabe, D.: Ontology Driven Dynamic Web Interface Generation. In: Proceedings of the 8th Int. Workshop on Web-Oriented Software Technologies (IWWOST 2009) in Conjunction with ICWE 2009, San Sebastian, Spain (June 2009)
6. Nunes, D.A., Schwabe, D.: Rapid prototyping of web applications combining domain specific languages and model driven design. In: Proc. 6th International Conference on Web Engineering (ICWE 2006), pp. pp. 153–160. ACM, New York (2006) ISBN 1-59593-352-2
7. Oren, Heitmann, B., Decker, S.: ActiveRDF: embedding Semantic Web data into object-oriented languages. *Journal of Web Semantics* 6(3), 191–202 (2008)
8. Rossi, G., Schwabe, D., Lyardet, F.: Web Application Models Are More than Conceptual Models. In: Procs. of the ER 1999, Paris, France, pp. 239–252. Springer, Heidelberg (1999)
9. Schwabe, D., Rossi, G.: An object-oriented approach to web-based application design. In: Theory and Practice of Object Systems (TAPOS), Special Issue on the Internet, vol. 4#4, pp. 207–225 (October 1998)
10. Silva de Moura, S., Schwabe, D.: Interface development for hypermedia applications in the semantic web. In: Proc. WebMedia and LA-Web, 2004, Ribeirão Preto, Brazil, pp. 106–113. IEEE Press, Los Alamitos (2004)
11. Thomas, D., Barry, B.M.: Model Driven Development: The Case for Domain Oriented Programming. In: Companion of the 18th OOPSLA, pp. 2–7. ACM Press, New York (2003)
12. Van Deursen, A., Klint, P., Visser, J.: Domain Specific Languages: An Annotated Bibliography, <http://homepages.cwi.nl/~arie/papers/dslbib/>
13. Vdovjak, R., Frasincar, F., Houben, G.J., Barna, P.: Engineering Semantic Web Information Systems in Hera. *Journal of Web Engineering* 2(1&2), 3–26 (2003)

A Quality Model for Mashups

Cinzia Cappiello¹, Florian Daniel², Agnes Koschmider³, Maristella Matera¹,
and Matteo Picozzi¹

¹ Politecnico di Milano, Dipartimento di Elettronica e Informazione
Via Ponzio 34/5, 20133 Milano, Italy
{cappiell,matera,picozzi}@elet.polimi.it

² University of Trento, Dept. of Information Engineering and Computer Science
Via Sommarive 5, 38123 Povo (TN), Italy
daniel@disi.unitn.it

³ University of Pretoria, Department of Computer Science
0002 Pretoria, South Africa
akoschmider@cs.up.ac.za

Abstract. Despite several years of mashup practice and research, it is still hard to find high-quality, useful mashups on the Web. While this can be partly ascribed to the low quality of the components used in the mashups or simply to the lack of suitable components, in this paper we argue that this is partly also due to the lack of suitable quality models for mashups themselves, helping developers to focus on the key aspects that affect mashup quality. Although apparently easy, we show that – if taken seriously – mashup development can be non-trivial and that it deserves an investigation that specializes current web quality assessment techniques, which are not able to cater for the specifics of mashups. In fact, we believe a mashup-specific quality model is needed.

1 Introduction

Mashups are Web applications that integrate heterogeneous resources at different levels of the application stack, i.e., at the data, application logic, and user interface (UI) level. The resources can be readily available on the Web, or they may have been purposely created in order to reach the goal of the mashup developer. For example, it could be necessary to implement a dedicated service to access company-internal data, in order to plot them on a Google map. Independently of where resources come from, the goal of mashups is to provide novel features by integrating resources in a value-adding manner, similar to service composition, which however only focuses on the application logic layer.

The interest in mashups has grown constantly over the last years, especially due to two main reasons. First, mashups and their lightweight composition approaches represent a *new opportunity* for companies to leverage on past investments in service-oriented software architectures and web services, as well as on the huge amount of public APIs available on the Web. In fact, the possibility to integrate web services with UIs finally caters for the development of complete applications, a task that service composition addressed only partially. Second, the

emergence of mashup tools, which aim to support mashup development without the need for programming skills, has moved the focus from developers to *end users*, and from product-oriented software development to consumer-oriented composition [1]. That is, mashup tools have refueled research on and investments in end user development.

While these reasons undoubtedly justify the extraordinary interest in mashups, we however observe that so far none of the envisioned benefits have been achieved. In fact, most of the mashups that can be found online are simple and relatively useless applications and, as such, far from applicable in the enterprise context. Moreover, mashup development is still a prerogative of skilled developers. If we look at the mashups themselves, we observe that their *quality* is simply low, a result that is worsened by the fact that apparently not only end users but even developers have difficulties in implementing high-quality mashups.

Given these premises, the aim of this paper is to understand how to systematically assess the quality of mashups and which are the aspects mashup developers should keep an eye on, in order to obtain mashups of better quality. In our previous research we investigated the quality of *mashup components* [2]. Specifically, we identified a set of quality dimensions and metrics that characterize the quality of mashup components, where each component was seen as a black box, a perspective that characterizes the reuse-centric nature of mashups. Starting from a study conducted on a set of existing mashups, in this paper we define a quality model for *mashups*.

The mashup assessment starts from the assumption that mashups, after all, are Web applications and, as such, can be evaluated by means of *traditional quality models*. We therefore analyzed about 70 mashups available on ProgrammableWeb.com, applying criteria and metrics for Web applications that traditionally focus on the perceived quality of Web applications, e.g., accessibility and usability. This study revealed that understanding the quality of mashups requires a quality model that takes into account the *specifics of mashups*. This led us to the definition of a quality model.

This paper is organized as follows. Next, we review existing techniques and quality models for software and Web application assessment. In Section 3, we apply some of the reviewed tools in order to perform our empirical assessment of mashup quality. In Section 4, we precisely define our idea of mashups and analyze their characteristics from a composition point of view. In Section 5, we derive a quality model that takes into account these characteristics, and we discuss a set of representative positive and negative examples. In Section 6 we conclude the paper.

2 Related Work

A quality model is a structured set of quality characteristics that, for instance, allows an assessor to evaluate Web resources from different perspectives. In the Web, first quality models focused on Web sites [3,4], then they focused on more complex Web applications [5,6]. Recently, quality models for Web 2.0 applications have emerged [7,8]. Yet, the Web 2.0 also introduced a novel role into the

quality assessment process, i.e., the end user feedback. In fact, it is nowadays common practice for Web 2.0 applications to be assessed via simple human judgments [9], without following any structured quality model. While user feedback is generally a rich source for quality assessment, the phenomenon of self-ratings by users cannot replace third-party assessments [10,11].

In particular, in the mashup context, traditional quality dimensions suggested for software engineering [12,13] and Web engineering [14,15] may be partly appropriate to measure the internal quality of a mashup (e.g., code readability), as well as its external quality in-use (e.g., usability). In Section 4.3, we will see that, a deeper and more reliable analysis can be achieved by taking into account the component-based nature of mashups. Instead, the W3C guidelines for *accessibility* [16] can be applied to Web sites and mashups alike, as accessibility is a rather technical concern.

Rio and Brito e Abreu [4] found out that most of the quality dimensions are domain-dependent, i.e., that there is a strong impact by the application domain on the usefulness of quality dimensions. In line with this finding, in [2] we started looking at the specifics of mashup components, leaving however open the problem of defining a comprehensive quality model for mashups.

Despite the lack of quality models for mashups, some existing approaches, for instance those focusing on the quality of Web 2.0 information [17,18], may contribute to the assessment of mashups. However, these approaches evaluate the input (resource streams) separately and not their final composition, which is a crucial factor for mashups on which we instead focus. In [19] we tried to partially fill this gap by studying how information quality propagates from components to mashups. Yet, non data-related aspects of internal quality and user interface have not been considered so far in the specific case of mashups.

3 Mashup Development: Quality Issues and Challenges

While in the previous sections we generically stated that “mashups are Web applications that integrate heterogeneous resources at different levels of the application stack, i.e., at the data, application logic, and user interface (UI) level,” for the purpose of our analysis we further refine this definition as follows:

Mashups are Web applications that integrate inside one web page *two or more* heterogeneous resources at different levels of the application stack, i.e., at the data, application logic, and UI level, *possibly putting them into communication among each other*.

The first reason for this refinement is that we specifically want to focus on mashups that have an own *UI* (to distinguish them, for example, from so-called data mashups as the ones created with Yahoo! Pipes) and that aim to provide added value by *integrating* a set of existing services or components, rather than coding something from scratch. That is, we want to emphasize the typical *component-based* nature of mashups and the resulting development paradigm of *composition*. In fact, mashup development is similar to other component-based

development paradigms, most prominently to web service composition, which focuses on the integration and *reuse* of existing web services. Mashups, however, extend this composition paradigm also toward data services and UI components, enabling the development of web applications that span all the three layers of the application stack.

If we neglect the static content possibly added by the mashup developer during integration (e.g., in the layout template of the mashup), we can identify two core independent aspects for which *quality* becomes an issue:

- *The components*: Since a mashup reuses data, application functionality, and/or UIs, the quality of these building blocks certainly influences the quality of the final mashup. The lower the quality of the chosen components, the lower the quality of the result of the composition. Even if a developer is aware of the low quality of a component, is it not always possible to choose a better component, e.g., because no other components implementing the same functionality are available.
- *The composition*: While it is usually not possible to improve the quality of third-party components, it is at the other hand relatively easy to further degrade the potential quality of a mashup, i.e., the maximum quality the mashup could have by integrating the same components, if the composition logic of the mashup is not well crafted. In fact, the composition logic is the only part of the mashup that is not reused and that needs to be implemented by the mashup developer, typically each time from scratch. Given the complexity of mashups and the usually high number of different technologies involved, this task is generally error-prone and hard to debug.

Additionally, composition is a complex task, which can be split into three independent sub-tasks, each with its very own quality concerns:

- *Data integration*: Integrating different components may require integrating their data, e.g., if one component is configured to provide input to another component. Doing so may require the re-formatting of data, the cleansing of data, joining or splitting data, and similar.
- *Service orchestration and UI synchronization*: Passing data from one component to another component also means setting up the necessary orchestration logic among services or UI synchronization logic among UI components, since services interactions are typically invocation-based and UIs event-based. Mixing the two integration paradigms is non-trivial in general.
- *Layout*: Finally, one of the most crucial aspects for the success of any application is its graphical appearance. The common practice in mashup development is to use HTML templates with place holders for the UI components to be integrated. Although apparently easy, developing good templates that are able to seamlessly host third-party UIs (possibly with customized CSS settings) is again far from easy.

Given these peculiarities, the *challenge* is to characterize the quality of mashups in a way that (i) captures the perceived quality of the mashup (so as to address

the user's perspective), but that also (ii) allows the mashup developer to act upon the composition logic, in order to settle possible quality issues.

4 Assessing Mashups Like Common Web Applications

In the following we describe a study that we conducted, in order to understand *how well current mashups perform in terms of existing quality criteria and how much useful information a mashup developer can obtain from the application of existing standards and quality evaluation tools for Web applications in general*. The hypothesis of the experiment was that, after all, mashups are nothing but regular Web applications.

We performed a systematic review of readily available quality assessment instruments, we chose four criteria for which there were automated assessment tools available (in order to eliminate as much as possible subjective evaluations):

- *Usability*: measures the ease of use of the mashup by the mashup users.
- *Readability*: measures how easy or difficult it is to read and understand the text rendered in the mashup.
- *Accessibility*: measures how well the mashup complies with the W3C web accessibility guidelines [16].
- *Performance*: measures the loading time of the mashup till all elements of the application are rendered in the page.

4.1 Setup of the study

In order to automatically assess mashups according to the selected criteria, we looked for tools that (i) are able to autonomously access and assess mashups starting from a common URL and (ii) are free. We then identified and selected the following instruments:

- *Site Analyzer* (<http://www.goingup.com/analyzer/>) for the assessment of W3C accessibility and usability guidelines, usability guidelines by J. Nielsen [20] and ISO standards. Altogether, *SiteAnalyzer*TM implements 70 automatically running tests. The evaluation results are shown on a 100-percentage scale, where results equal or higher than 75% indicate a *very good* conformance, result between 75% and 65% a *moderate* conformance and all below 65% represents an insufficient conformance.
- *Juicy Studio* (<http://juicystudio.com/services/readability.php>) for the assessment of readability using the Flesch Reading Ease algorithm, which inspects the average number of words used per sentence and average number of syllables per word in order to compute a number that rates the text on a 100-point scale. The higher the score, the easier the resource to understand. The recommended score for an object is approximately 60 to 70.
- *Pingdom* (<http://www.pingdom.com/>) for measuring mashup loading times. The tool loads all objects (e.g., images, CSS, JavaScripts, RSS, Flash and frames/iframes) and shows the total load time in seconds and visually with time bars.

In order to select the set of candidate mashups for our study, we went to the largest registry of mashups available online, i.e., the ProgrammableWeb.com Web site, which allows mashup developers from all over the world to link their own mashups and to provide some useful meta-data about them. Developer-provided meta-data are, for instance, the *publication date* of the mashup or the set of *APIs* used by the mashups, while the site also features user-provided *ratings* for mashups and the *number of users* that accessed it. Mashups are not hosted on ProgrammableWeb.com, but linked.

Out of the population of 5347 mashups (at the time of the experiment) we randomly selected 224 candidate mashups. We used a *simple random sample* (SRS) technique, a special case of a random sample [21] in which each unit of the population has an equal chance of being selected for the sample. We did not, for instance, focus on the “top 100” mashups, in order to have an as representative as possible sample without bias toward high quality.

Unfortunately, the links of 87 mashups were broken, and 22 mashups could not be processed by the analysis tools. Also, 43 pointed pages could not be considered proper mashups, being them simple Web pages not including any external API. So, the final sample for the evaluation was composed of 68 mashups.

The website of the *Site Analyzer* tool publishes information about the usability and accessibility of top rated web pages. We selected 74 web sites out of 100 (the remaining 26 web sites are variants of each other or duplicates with no difference in rating) and analyzed them according to the selected four quality criteria (usability, readability, accessibility and performance).

Subsequently, we compared the measures achieved for those web sites with the measures of the mashups in order to assess the validity of the values for mashups.

4.2 Results

Before the actual quality assessment, we tried to understand whether the meta-data available in ProgrammableWeb.com are somehow correlated to the mashup quality and as such can be considered quality indicators. The performed statistical tests were not able to identify any dependency among the number of users and the average rating of mashups, nor among the publication date and the number of users. This means that the meta-data cannot be used to obtain indications on the quality of mashups.

Table 1, instead, shows the results of the evaluation of the four quality criteria for the five “best” and the five “worst” mashups in terms of usability.

In general, the tool analysis indicates that about 25% of the mashups are of very good usability and accessibility. The average readability degree is close to 60, which indicates an “easy” reading, and the average loading time is close to 5 seconds, which we can consider at the limit of acceptability.

An in-depth analysis of the mashups with very good usability and accessibility (17 mashups) revealed a Spearman’s correlation coefficient indicating a high correlation between usability and accessibility (0.855 and p-value < 0.005). This means that usability and accessibility are coherent. The average readability

Table 1. Evaluation results for the five “best” and five “worst” mashups

Rating	Mashup	Usability	Readability	Accessibility	Performance
1	A Paris Hilton video site	83.9%	65.2	85.5%	3.1 sec.
2	Sad Statements	81.4%	35.2	80.5%	5.1 sec.
3	ShareMyRoutes.com	79.8%	62.9	78.4%	2.8 sec.
4	DiveCenters.net	79.5%	75.0	73.3%	1.4 sec.
5	Cursebird	79.1%	78.1	79.3%	2.1 sec.
...
64	CityTagz	65.0%	53.3	64.2%	3.2 sec.
65	Blue Home Finder	64.9%	77.7	63.9%	7.3 sec.
66	Gaiagi Driver - 3D Driving Simulator	64.8%	53.9	64.6%	6.3 sec.
67	2008 Beijing Olympics Torch Relay Path	62.2%	10.5	58.7%	2.0 sec.
68	Tidespy: Tide Charts with Best Fishing Times	61.2%	58.2	64.3%	5.7 sec.

degree of these 17 mashups was 60.05 and their loading time was 5.33 seconds. In contrast to that, the mashups with very low usability and accessibility had a load time of 3.70 seconds and an average readability degree of 42.66.

Next, in order to understand how mashups with low quality could be improved, i.e., to provide mashup developers with suitable guidelines and to help them improve the mashup, we analyzed the error and warning messages produced by *Site Analyzer*, which specifically focuses on usability and accessibility. In Table 2 we report the five most recurrent warnings and suggestions we collected during our analysis, along with their description.

4.3 Analysis of Results

In order to understand whether the results of the automatic analysis were reliable, we conducted five independent evaluations by manually inspecting the same mashups and assigning them with a score expressing the mashup quality and a comment justifying the score. We then compared the results of the two evaluation sessions and discovered that in several cases the two assessments diverged.

We immediately identified some inconsistencies. There are indeed some examples that show that the tools were not able to capture some peculiar features of mashups and therefore under- or over-estimated their quality. Let us give an example: Figure 1 shows the Gaiagi 3D Driver mashup, which provides a 3D driving simulation achieved through different map APIs. The user can search for a route, and the application simulates the journey thanks to the Google Earth API and the Google Street View API. A Google Map and a Bing map show a traditional map representation. A text description then shows the route indications. This mashup is very rich from the functionality point of view. Although the included APIs have similar purposes (they all provide map-based services), their orchestration offers a valuable, multi-faceted, still integrated view over route indications. The quality of this mashup, as assessed through the automatic tools, is low: i) the accessibility test obviously failed - a map cannot be interpreted by any accessibility tool (e.g., screen readers); ii) although the usability scores were

Table 2. Most recurrent warnings and suggestions for analyzed mashups

Warning	%	Description	Comment
Visited Links	68.69	Marking previously visited hyperlinks helps users to identify which pages of the website have been previously read. In other words, according to Nielsen the navigation interface should answer three essential questions: "Where am I?", "Where have I been?" and "Where can I go?	Mashups typically consist of single pages only. So, the links inside the mashups mostly serve to interact with the single page and its application features, e.g., to show the location of a housing offer on a map, and less to navigate through a complex hypertext structure. Users do not get lost in one-page mashups.
Privacy Policy	64.34	The owner of the website uses the Privacy Policy to inform users how personal data will be treated and how data protection regulations will be observed.	Given that mashups integrate data from a variety of sources, it may be important to explicitly state where data come from and how they are used.
Valid XHTML	54.78	Implementing valid (X)HTML markup may improve the quality of the rendering inside the client browser.	Due to their component-based nature, it is hard in mashups to fine-tune the markup code of third-party components.
Labelling links on mouse-over	53.04	To allow users a good navigation of your website, it is necessary that links can be identified and that they can be highlighted in color with a mouse-over.	Again, this is a feature that needs to be implemented by the developers of the components. Adding mouse-over or similar effects are hard to overlay afterwards.
Resizable Fonts	53.04	Not all visually impaired people make use of technical support to navigate on the internet. The feature to increase fonts is an important element to ensure that all information of the website can be read by all users. Even people who are not visually impaired need to increase the font sometimes.	Increasing or decreasing the fonts inside a mashup may reveal very different behavior from the integrated components. While some of them may implement resizable fonts, others may completely neglect the issue, revealing it only when readability instead needs to be increased.
Suggestion	%	Description	Comment
Sitemap	90.43	Sitemaps allow users to have a quick and complete overview of the website.	As already pointed out, mashups typically consist of one page only. So, a sitemap is actually not needed.
Printer Friendliness	84.34	Web users often want to print the content of a page. Neat printing of the content (e.g., without having the borders cut off) can be achieved by implementing a print function and also by including stylesheets which have been optimized for printing.	While this is a capability that is definitely desirable also for mashups, so far mashups have been focusing more on outputs that are either data (e.g., Yahoo! Pipes) or interactive UIs, which are not easy to print.
Table Summary	61.73	The TABLE element takes an optional SUMMARY attribute to describe the purpose and/or structure of the table. The overview provided by the SUMMARY attribute is particularly helpful to users of non-visual browsers.	If tables are used for the layout of the mashup, a summary could indeed help impaired users to understand the structure of the mashup. Tables inside the components are outside the control of the mashup developer.
Image with Missing Width or Height	59.13	Missing height and width size of images impacts the performance of the website. A precise definition of height and width allows a quicker download of the website, as the browser recognizes the space needed for the image and can leave it empty.	This suggestion applies equally to mashups. Given the typically dense integration of contents (components) inside a mashup page, correct formatting may be crucial for successful rendering.
Table Header	46.08	Table headers should be recognizable as such, since they perform a descriptive task.	The same holds for mashups. Again, tables of the layout template are under the control of the mashup developer, tables inside components not.

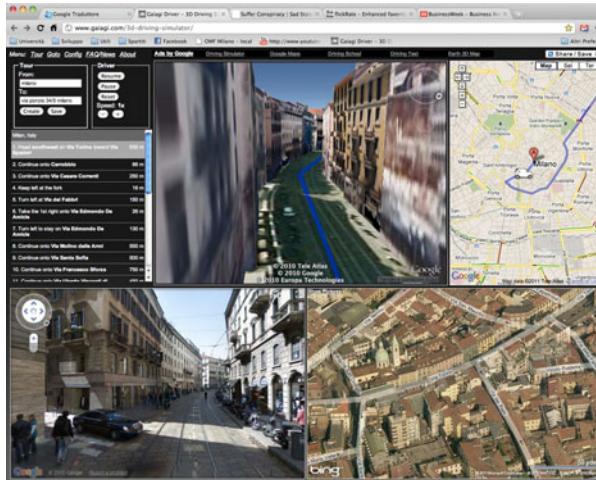


Fig. 1. A mashup integrating different map-based components (<http://www.gaiagi.com/3d-driving-simulator/>)

moderate (Readability = 53.9; Usability = 64.6%) the mashup was ranked at position 66 out of the 68 mashups.

Conversely, according to the five manual inspections, this mashup was the top ranked: it is easy to use, effective from the point of view of the offered data and functionality, highly interactive and very well orchestrated. This therefore shows that the automatic tools do not capture at all this good quality. One reason is that some criteria that are cornerstone for traditional Web applications, e.g., the richness of links and intra-page navigation, and the readability of text, do not necessarily apply to mashups. For example, map-based components, as the ones used in the example of Figure 1, do not necessarily show text, rather they visually render some points on the map space. Their effectiveness is however still high and proved in several contexts.

Another reason for the observed discrepancy is that, as better discussed in Section 5, important ingredients in mashups are the richness and suitability of components, as well as the way they synchronize. Mashups should indeed be able to offer an integrated view over the different domains deriving from the integrated services. This characterization is not obvious; in other words there is no general consensus on what a mashup is and what not. This is also confirmed by the fact that 29% of the mashups randomly selected from ProgrammableWeb.com included one single API (in most cases a map), without offering any real integration among multiple components. However, we strongly believe that a discriminant factor, which allows one to identify the actual value of a mashup, is the presence of multiple components and a reasonable number of coupling mechanisms giving place to a real integration. These aspects cannot be captured by automatic tools that therefore provide too generic feedback not reflecting at all the salient characteristics of mashups.

The shortcomings of state-of-the-art tools are however not limited to the computation of quality metrics only. In fact, as the comments in Table 2 show, also the feedback provided to developers in terms of warning messages and suggestions is only hardly applicable in the context of mashups. It is generally hard to change any of the hypertext characteristics of the components used inside a mashup, while it is definitely possible to fine-tune the composition logic of the mashup. Yet, this latter aspect is not supported in terms of suitable warnings or suggestions.

5 The Mashup Quality Model

As the previous analysis shows, existing quality dimensions are not totally suitable for evaluating the quality of mashups. In this section, we therefore aim at identifying a set of properties that are able to capture the capability of a mashup to enable users to access specific content and functions with easiness, effectiveness, safety, and satisfaction. As we have seen in Section 2, traditional Web pages are usually evaluated by focusing on the value of the provided information and the suitability of the presentation layer [22]. On the basis of the definition provided in Section 3, mashup quality also depends on the quality of the components involved in the mashup and on the quality of the composition in terms of data, functionalities, and interface integration. Therefore, for mashups, traditional dimensions should be revised and enriched with dimensions addressing the added-value that these applications provide through the integration of different components.

We propose a model that classifies quality dimensions into three categories, i.e., *Data Quality*, *Presentation Quality*, and *Composition Quality*, where the latter category is very peculiar for mashups since it focuses on the way components interact among each other and measures the utility of the composition. In order to better define these three categories (in the rest of this section), we first characterize the mashup composition task as follows:

- Mashups are developed in order to offer a set of application features FS , and consequently retrieve and give access to a set of data that we call the Data Set DS . An *application feature* is a functional capability, i.e., an operation that can be executed by the users or automatically performed on behalf of the users (e.g., due to synchronization between components).
- A mashups can be associated with the set of components $C = \{c_1, \dots, c_k\}$ used to create it. Each component c_i , $1 < i < k$, has its own set of application features FS_i and data set DS_i . To fulfill the mashup requirements, smaller portions SFS_i and related SDS_i are sufficient. They are called situational features and data sets.

Figure 2 sketches the quality aspects and dimensions addressed by our mashup quality model, which we discuss next in more detail.

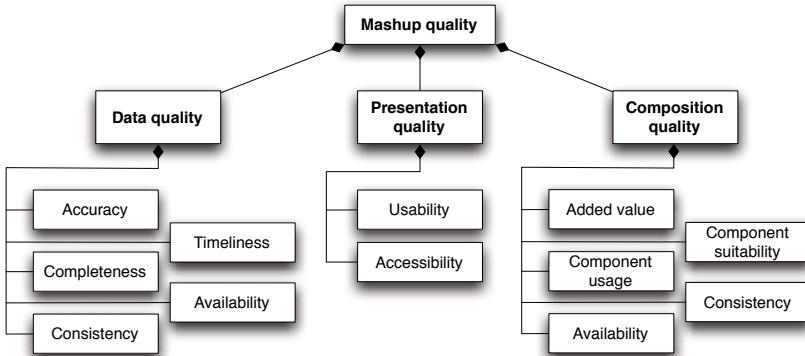


Fig. 2. The dimensions of the quality model for mashups

5.1 Data Quality

As defined in [19], the mashup quality strongly depends on the information that the integration of different components is able to provide. Assessing the quality of a mashup therefore requires understanding both the quality of components and the effect that the composition has on the overall quality of the final mashup. In [19], we have defined an information quality model for mashups from which we reuse the following set of dimensions relevant for mashup applications:

- *Accuracy*: refers to the probability that mashup data are correct. This is calculated by considering the probability that an error occurs in the data sets SDS_1, \dots, SDS_k of the components used in the mashup. This probability can be estimated by looking at the usage history of a component and considering all the types of occurred errors (e.g., typos, wrong representation).
- *Situational Completeness*: defines the degree with which DS is able to provide the desired amount of information. The evaluation of such dimension starts from considering the amount of data contained in SDS_1, \dots, SDS_k and comparing them to the amount of data that ideally should be accessed through the mashup.
- *Timeliness*: provides information about the freshness of the available data sets. The timeliness of the mashup results from the aggregation of the timeliness of the different components where the aggregation function can be minimum, average, or maximum.
- *Consistency*: addresses situations in which the mashed up components have data sets that conflict with each other, leading to inconsistency in the data shown in the final mashup.
- *Availability*: is the likelihood for the mashup to be able to provide any data, that is, in order for a mashup to be available at least one of its components must be available.

It is worth to point out that the way in which components interact among them influences the information quality assessment, especially for accuracy and

situational completeness dimensions. Timeliness and availability are instead computed as simple aggregation of the respective characteristics of the individual component data sets.

5.2 Presentation Quality

Presentation Quality is traditionally decomposed into *usability* and *accessibility*:

- *Usability*: also in the mashup context, usability addresses some traditional dimensions, such as *orientation*, *users control*, *predictability*, *layout consistency*. For example, the features provided by a mashup should be invoked through commands that are easy to identify and understand. The users should also easily understand the effect of a feature when it is invoked. In particular, in a mashup that synchronizes multiple components, the effects of the propagation of a command over the different components should be visible to the user.

An important usability attribute is *learnability*, which relates to “the features of an interactive system that allow novice users to use it initially, and then to attain a maximum level of performance” [23]. Learnability should be privileged in mashups, even though this application are very often simple: the mashup features should be visible enough and the corresponding commands should be self-expressive so that even naive users can easily master the mashup execution.

Finally, a major role is played by *layout consistency*, which provides the “glue” through which even heterogenous components result to be a whole. However, some other usability attributes usually adopted for traditional Web applications are not applicable in the evaluation of mashup quality. First of all, even though future development practices would lead to the creation of multi-page mashups [24], current mashups are still very simple applications from the point of view of their hypertext structure – very often they are made of a single page. Therefore, criteria such as navigability and richness of links, or any other criteria addressing the hypertext structure, which are generally considered relevant for the quality of Web applications [25], have a lower impact over the mashup quality.

Moreover, mashup applications rarely contain long texts and the assessment of all the quality dimensions that are commonly used for the evaluation of Web content, such as *readability*, *cohesion* or *coherence*, might provide irrelevant results.

- *Accessibility*: it addresses properties of graphic page design, such as those defined by the W3C Web Accessibility Initiative (WAI) [26]. Accessibility criteria do not need to be specialized for mashups. Rules for enabling access by any user and any device should indeed be taken into account when defining the layout of the mashup, also smoothing some layout setting of single components in case they violate any accessibility rule.

5.3 Composition Quality

As stated above, composition quality aims at evaluating the component orchestration and the suitability of the mashup for the provision of the desired features. The composition quality is decomposed into *added value*, *component suitability*, *component usage*, *consistency* and *mashup availability*.

- *Added value*: the *added value* of the composition can be related to the amount of provided features and/or offered data. The composition provides an added value if, for each component c_i , $SFS_i \subset FS \vee SDS_i \subset DS$, i.e., the amount of features/data offered by the mashup is greater than the amount of features/data offered by the single components. While the previous one defines the minimum condition ensuring that some added value is provided by the composition, the goal to be reached is that $\bigcup_i SFS_i \subset FS \vee \bigcup_i SDS_i \subset DS$. This also reflects the principle that information becomes more valuable when it can be compared and combined with other information [27].
We can quantify the added value along a scale that ranges from the case in which a mashup gives simply the opportunity to render data coming from different sources (without any attempt to integrate them) to the case in which additional features and data are provided by an adequate integration. For example, a mashup as *dailymashup.com* provides a low added value since its single page offers a very poorly integrated view on some selected news (taken from Yahoo!news) and, in a totally unaligned fashion, also on the top-ranked last 24 hours photos from Flickr. More added value would be offered if the mashup components were synchronized. To provide an added value, mashups must offer to the users additional features or data, as for example the mashup published on *www.bluehomefinder.com*, where an advanced service for finding houses offers the localization of houses on a map plus other features that allow the users to perform operations of filtering and selection.
- *Component suitability*: it refers to the appropriateness of the component features and data with respect to the goal that the mashup is supposed to support. For example, a mashup that aims at providing addresses of the nearby restaurants with respect to the current user location should be effectively built based on map-based components. In fact, a simple list of textual addresses could not appropriately support those users that are not acquainted with the geographical area.
- *Component usage*: it may happen that, even though a component is very rich from the point of view of data and functionality, it is improperly used within a composition. For example, the Google Maps API offers several operations, such as geocoding, directions, street view, and elevation. Let us consider that a mashup developer decides to just use the street view feature. This choice is not reasonable if the user goal is to get oriented within a geographical area: the street view just offers a local and realistic view of a selected point of interest, while it does not provide a larger view of the area in which the point is located.
- *Consistency*: poor quality compositions can also be caused by *inconsistencies* at the orchestration level. In fact, the composition of two components

is feasible if the two linked operations are compatible from only a syntactic perspective even if they are incompatible from a semantic perspective. In this way, the composition can produce inaccurate results. For example, the composition between an operation that provides an address and a map service is feasible since the input-output parameters are strings. However, if the address is incomplete and contains only information about the city in which the point of interest is located, this will work properly but it will not show the desired information.

- *Availability*: the degree in which the mashup can be properly accessed during a given time interval. It depends on the availability of the components and on their role in the composition.

6 Conclusion and Future Work

To the best of our knowledge, this paper represents the first attempt to define a quality model for mashups. Although many so-called “quality-aware” or “quality-based” composition approaches (mostly for web service composition) have been proposed so far, a holistic approach to quality assessment of the final output of the composition task – especially for mashups – was still missing.

Driven by the results of our study, where we evaluated a huge sample of mashups, we took a better look at the peculiarities of mashups and understood that, after all, mashup developers do not have full control over *what* they integrate (the components) but only over *how* they integrate (the composition logic). Accordingly, we defined a quality model that emphasizes this component-based nature of mashups and especially focuses on composition quality.

We recognize that many of the quality dimensions introduced in this paper are not easy to turn into operative metrics and to automatically assess, yet we also recognize that quality assessment to a large degree will always be a *qualitative* process. In [19] we started proposing more concrete metrics for data quality; usability and accessibility are already supported to a large extent with (semi-) automated evaluation metrics; the next challenge is supporting composition quality with suitable metrics. This is part of our future research.

Acknowledgments. This work was partially supported by funds from the European Commission (project OMELETTE, contract no. 257635).

References

1. Nestler, T.: Towards a mashup-driven end-user programming of soa-based applications. In: iiWAS, pp. 551–554. ACM, New York (2008)
2. Cappiello, C., Daniel, F., Matera, M.: A quality model for mashup components. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 236–250. Springer, Heidelberg (2009)
3. Ivory, M.Y., Megraw, R.: Evolution of web site design patterns. ACM Transactions on Information Systems 23, 463–497 (2005)
4. Rio, A., e Abreu, F.B.: Websites quality: Does it depend on the application domain? In: Proc. of QUATIC, pp. 493–498. IEEE Computer Society, Los Alamitos (2010)

5. Olsina, L., Rossi, G.: Measuring web application quality with webqem. *IEEE Multimedia* 9, 20–29 (2002)
6. Mavromoustakos, S., Andreou, A.S.: WAQE: a web application quality evaluation model. *Int. J. Web Eng. Technol.* 3, 96–120 (2007)
7. Olsina, L., Sassano, R., Mich, L.: Specifying quality requirements for the web 2.0 applications. In: Proc. of IWWOST 2008, pp. 50–56 (2008)
8. Knap, T.T., Mlýnková, I.: Quality assessment social networks: A novel approach for assessing the quality of information on the web. In: Proc. of QDB, pp. 1–10. ACM Press, New York (2010)
9. Varlamis, I.: Quality of content in web 2.0 applications. In: Setchi, R., Jordanov, I., Howlett, R.J., Jain, L.C. (eds.) *KES 2010. LNCS*, vol. 6278, pp. 33–42. Springer, Heidelberg (2010)
10. Brooks, C.H., Montanez, N.: Improved Annotation of the Blogosphere via Auto-tagging. In: Proc. of WWW, pp. 625–632. ACM Press, Edinburgh (2006)
11. Heymann, P., Koutrika, G., Garcia-Molina, H.: Can Social Bookmarking Improve Web Search? In: Proc. of the WSDM, pp. 195–206. ACM Press, New York (2008)
12. Kan, S.H.: Metrics and Models in Software Quality Engineering. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
13. ISO/IEC: ISO/IEC 9126-1 Software Engineering. Product Quality - Part 1: Quality model (2001)
14. Calero, C., Ruiz, J., Piattini, M.: Classifying web metrics using the web quality model. *Online Information Review* 29, 227–248 (2005)
15. Olsina, L., Covella, G., Rossi, G.: Web quality. In: Mendes, E., Mosley, N. (eds.) *Web Engineering: Theory and Practice of Metrics and Measurement for Web Development*, pp. 109–142. Springer, Heidelberg (2005)
16. W3C: Web Content Accessibility Guidelines (WCAG) 2.0 (2008)
17. Abramowicz, W., Hofman, R., Suryn, W., Zyskowski, D.: Square based web services quality model. *Information Systems Journal* 1, 1–9 (2008)
18. Zhang, H., Zhao, Z., Sivasothy, S., Huang, C., Crespi, N.: Quality-assured and sociality-enriched multimedia mobile mashup. *EURASIP J. Wirel. Commun. Netw.*, 11:1–11:13 (2010)
19. Cappiello, C., Daniel, F., Matera, M., Pautasso, C.: Information quality in mashups. *IEEE Internet Computing* 14, 14–22 (2010)
20. Nielsen, J.: The usability engineering life cycle. *Computer* 25, 12–22 (1992)
21. Berger, V.W., Zhang, J.: Simple Random Sampling. *Encyclopedia of Statistics in Behavioral Science* (2005)
22. Aladwani, A.M., Palvia, P.C.: Developing and validating an instrument for measuring user-perceived web quality. *Inf. Manage.* 39, 467–476 (2002)
23. Dix, A., Finlay, J., Abowd, G.D., Beale, R.: *Human Computer Interaction*, 3rd edn. Pearson, Harlow (2003)
24. Daniel, F., Soi, S., Tranquillini, S., Casati, F., Heng, C., Yan, L.: From people to services to ui: Distributed orchestration of user interfaces. In: Hull, R., Mendling, J., Tai, S. (eds.) *BPM 2010. LNCS*, vol. 6336, pp. 310–326. Springer, Heidelberg (2010)
25. Ceri, S., Matera, M., Rizzo, F., Demaldé, V.: Designing data-intensive web applications for content accessibility using web marts. *Commun. ACM* 50, 55–61 (2007)
26. Consortium, W.: Wai guidelines and techniques. Technical report (2007), <http://www.w3.org/WAI/guid-tech.html>
27. Measuring The Value Of Information: An Asset Valuation Approach. In: Proc. of ECIS 1999 (1999)

DashMash: A Mashup Environment for End User Development

Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Gabriele Sprega,
Donato Barbagallo, and Chiara Francalanci

Politecnico di Milano - DEI
P.zza Leonardo da Vinci, 32 - 20133 - Milano - Italy
`{name.surname}@polimi.it`

Abstract. Web mashups are a new generation of applications based on the “composition” of ready-to-use services. In different contexts, ranging from the consumer Web to Enterprise systems, the potential of this new technology is to make users evolve from passive receivers of applications to actors actively involved in the “creation of innovation”. Enabling end users to self-define applications that satisfy their situational needs is emerging as an important new requirement. In this paper, we address the current lack of lightweight development processes and environments and discuss models, methods, and technologies that can make mashups a technology for end user development.

Keywords: mashups, service-based dashboards, end user development.

1 Introduction

Web mashups are a new generation of tools that support the “composition” of applications starting from services and contents oftentimes provided by third parties and made available on the Web. Mashups were initially conceived in the context of the consumer Web, as a means for users to create their own applications starting from public programmable APIs or contents taken from Web pages. However, the vision is towards the development of more critical applications, for example the so-called *enterprise mashups* [10], a porting of current mashup approaches to company intranets. The potential flexibility of mashup environments can help people help themselves [20], by enabling the on-demand composition of the functionalities that they need. Mashups are therefore emerging as a technology for the creation of innovative solutions, able to respond to the different problems that arise daily in the enterprise context, as well as in any other context where flexibility and task variability are dominant requirements.

Given the previous premises, the need arises to provide mashup environments where the end users (i.e., the main actors of this new development process) can easily and quickly self-construct their applications without necessarily mastering the technical features related to service invocation and integration. This is true in every context - not only in the Enterprise: a “culture of participation” [8], in which users evolve from passive consumers of applications to active co-creators of new ideas, knowledge, and products, is indeed more and more gaining momentum

1.1 Contributions and Paper Outline

What makes mashups different from plain Web service compositions is their potential as tools through which end users are empowered to develop their own applications. However, this potential is rarely exploited. So far the research on mashups has focused on enabling technologies and standards, with little attention on easing the mashup development process - in many cases mashup creation still involves the manual programming of the service integration. Some recent user-centric studies [14] also found that, although the most prominent mashup platforms (e.g., Yahoo!Pipes, Dapper or Intel Mash Maker) simplify the mashup development, they are still difficult to use by non technical users. In this paper, we try to respond to the need of easing the mashup development, and propose a Web platform, *DashMash*, that allows end users to develop their own mashups making use of an intelligible paradigm that abstracts from technical variables. In particular:

1. Through a case study, we provide a scenario in which general mashup composition principles can be applied to the construction of applications targeting the experts (e.g., analysts and decision makers) of a given domain (Section 2). Based on this scenario, we outline relevant factors supporting end user development. First of all, the importance of a *lightweight development process*, in which mashup composition paradigms are embedded in usable visual environments hiding the complexity of the composition languages actually managing the execution of the mashup.
2. We present a runtime architecture supporting the lightweight development processes, which increases the user's control over the mashup composition process (Section 3). This is possible thanks to (i) an instant execution support, based on the “on the fly” interpretation of the user composition actions and the immediate execution of the mashup composition in a WYSIWYG (What You See Is What You Get) manner (Section 3.2), and (ii) the automatic generation of descriptive models for the results of the users composition actions (Section 3.3), which then drive the execution of the mashup.
3. We promote the adoption of *recommendation mechanisms* that take into account quality variables to help end users select data sources and mashup components and composition patterns (Section 3.3). This is enabled by the enrichment of descriptive models with annotations specifying also non-functional user requirements.
4. Based on the results of a usability experiment, we discuss the effectiveness of our approach from an end user perspective (Section 4).

2 Case Study: Mashups for Sentiment Analysis

In the context of a project funded by the Comune di Milano (Milan Municipality), we have worked on the construction of a Web platform through which end users can construct their dashboards for *sentiment analysis*¹. Sentiment analysis

¹ A demo is available at

<http://home.dei.polimi.it/cappiell/demo/DemoDashMash.mov>



Fig. 1. Example of mashup composition for a sentiment analysis dashboard

focuses on understanding market trends starting from the unsolicited feedback provided by users comments published on the Web. Our project focuses on the design of an engine in charge of the automatic extraction of sentiment indicators summarizing the opinions contained in user generated contents [1], and on the provision of a Web environment where analysts can self-construct their analyses.

After preliminary attempts with a traditional static dashboard, we realized that end users could benefit from the ability to compose their analysis flexibly, playing in variable ways with sentiment indicators, and also complementing such indicators with “generic” external Web resources. This latter feature would indeed help them to improve their analyses, by interpreting sentiment indicators with a view on the events that cause trends and behaviors.

Figure 1 shows an example of use of the Web front-end of our platform, DashDash. A left hand menu presents the list of components, which for the sentiment analysis domain are *data sources* that materialize contents extracted from community sites, several types of *filters*, a multiplicity of *viewers* to visualize data, which are both open APIs, e.g, the Google APIs for maps and charts, and ad-hoc developed services², and utility open API/services, such as RSS feeds and calendars. Components can be mashed up by moving their corresponding icons into the so-called *workspaces*. Each workspace is associated with a data set

² Several charts offering advanced data visualizations have been developed using the Highcharts JS library (<http://www.highcharts.com/>).

resulting from the integration of data sources and filters, and renders this data set according to the visualizations offered by the selected viewers.

In the mashup shown in Figure 1a) the user has selected two data sources storing users comments extracted from two well-known social applications, Twitter and TripAdvisor. A filter is applied to select the only comments from users that are considered opinion leaders, so-called *influencers*. Influencers data are visualized through a *list viewer*, which is integrated with *Google Maps* to show the influencers locations. A further synchronization with another map and another list viewer allows one to see the original posts of each influencer, as well as the geo-localization of their posts, if available.

Users can iteratively modify the composition, by adding or dropping components. Changes are enacted at real time and the effect are immediately shown. They can also access a visual description of the status of the current composition (see Figure 1b), and easily modify sources, filters, viewers or even configuration properties of each single component.

Although the system automatically includes some default bindings to ensure basic inter-component synchronization, through simple dialog boxes the users can create new service combinations resulting into synchronized behaviors. For example, starting from the mashup shown in Figure 1a, the dialog box presented in Figure 1c allows the user to add a pie-chart viewer to show the distribution of influencers comments along different topics, and set a coupling so that a click on a pie slice contextualizes the analysis offered by the map viewer to that selected portion of data. Also, based on compatibility rules and quality criteria, the system provides suggestions about other candidate components to extend the mashup or replace existing components.

As shown by the previous example, the DashMash environment is characterized by factors that try to alleviate as much as possible the end users during the mashup composition task [14,8]:

- *Abstraction from technical details*: the representation of services as visual objects that abstract from technical details (e.g., their programmatic interface), the immediate feedback on composition action, and the immediate execution of the resulting mashup to reveal the service look&feel, help users realize the service functionality and the effect that the service has on the overall composition. In the end, users are asked to manipulate (e.g., add, remove or modify) visual objects focusing on the service visualization properties rather than technical details of service and composition logics. As also confirmed by a user-based experimentation (see Section 4), this increases user satisfaction and the user-perceived control over the composition process.
- *Composition support*: the composition task is guided in multiple ways. On the one hand, starting from components' descriptive models, the composition engine is able to infer and automatically create *default bindings* between the services that the users add to the composition. Compatibility rules and quality criteria are then adopted to “rank” available components and provide users with suggestions about *additional components* and *custom bindings*.

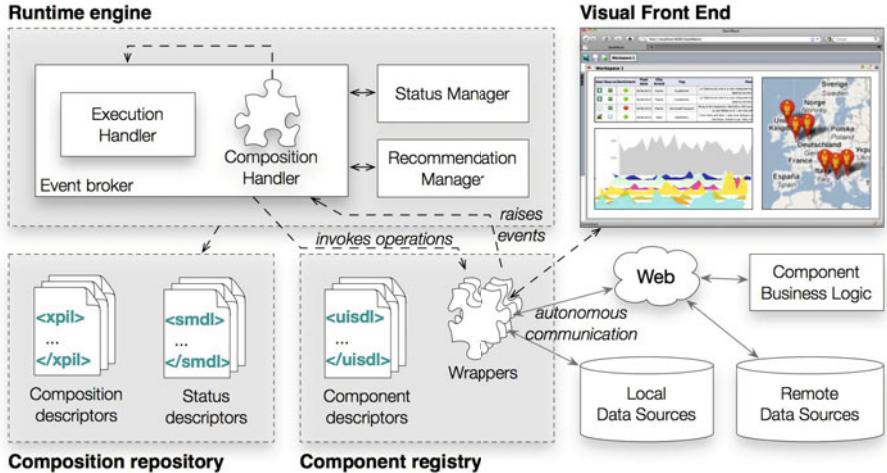


Fig. 2. Organization of the DashDash architecture

- *Continuous monitoring*: users are provided with mechanisms that allow them to understand the current state of the composition and to explore options about how to complete or extend the current composition.

The following sections are devoted to clarifying the modelling abstractions and the architectural features that allowed us to implement the previous requirements in DashDash.

3 The DashDash Platform

As represented in Figure 2, the organization of DashDash is centered around a lightweight paradigm in which the orchestration of registered services, the so called *Components*, is handled by an intermediary framework, in charge of managing both the *definition* of the mashup composition and the *execution* of the composition itself. Different from the majority of mashup platforms, where mashup design is separate from mashup execution, in DashDash the two phases strictly interweave. The result is that composition actions are automatically translated into models describing the composition; these models are immediately executed. Users are therefore able to *interactively* and *iteratively* define and try their composition, without being forced to manage complicated languages or even ad-hoc visual notations.

The current implementation of the DashDash runtime engine consists of a client-side (JavaScript) application that supports an *event-driven* execution paradigm. As represented in Figure 2, an *Event Broker* intercepts events, which can refer to users and system actions occurring during mashup execution (e.g., the click on a pie-chart slice which cause a change in a map), and to the dynamic definition of the composition (e.g., the drag&drop of a component icon into a workspace). The Event Broker then dispatches the events to the modules

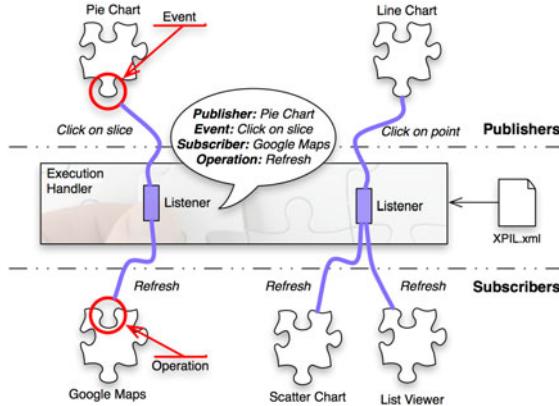


Fig. 3. Event-driven paradigm for service coupling definition and mashup execution

in charge of their handling, based on models and mechanisms that we explain in the rest of this section.

3.1 Event-Driven Execution

Events occurring during mashup execution are managed by an *Execution Handler* based on a *publish-subscribe model* addressing component integration at presentation level [21]: *events* generated from the user interaction with one mashup component (e.g., the selection of a slice in a pie chart) can be mapped to *operations* of one or more components that subscribe to such events (e.g., the visualization of details of the selected data in a scatter plot).

As represented in Figure 3, service couplings are expressed through the definition of the so-called *listeners* in a *composition model* expressed according to an XML-based language, *XPL* [21]. During mashup execution, each component keeps running according to its own application logic, within the scope defined by an HTML `<div>`. As illustrated in Figure 3, as soon as events occur, the involved components publish them. Based on the defined listeners, the Execution Handler then notifies the subscribed components, if any, and triggers the execution of their corresponding operations.

This composition and execution logics requires each component to be characterized by a high-level model expressing the *events* that the component can generate, the *operations* that enable other components to modify its internal state, and the binding with the actual service/API, so that operations can be invoked. Therefore, as illustrated in Figure 2, for each registered component the component registry stores a *descriptor*, expressed according to the *UISDL* language [21], and a *wrapper*, in charge of raising events and invoking operations. This component model provides a uniform paradigm to coordinate the mashup composition and execution, which obviates the heterogeneity of service standards and formats, also hiding the intrinsic complexity of services and composition.

3.2 Managing Composition

The *Composition Handler* manages composition events. In particular, as better explained in Section 3.3, it automatically translates the addition of a component into new *default listeners* and creates or updates (if already existing) the current composition model accordingly. The Composition Handler in turns dispatches composition events to the *Status Manager*, a module in charge of maintaining a *mashup state* representation. Combined with the composition model, the mashup state is useful to recover a previously defined mashup for a later execution, but especially to let users monitor their composition and modify it on the fly. State variables relate to default or specific parameter values (e.g., the value of a parameter for querying a data source), to layout properties (e.g., the colors used to show values on a chart) or to any other property that the user can set to control the component data and appearance. Composition events are also dispatched to the *Recommendation Manager*, a module in charge of providing suggestions about further components to select for extending/completing a composition (see Section 3.3). After terminating the handling of such events, the mashup composition is reloaded and immediately rendered into the workspace. The mashup is then executed according to the event-driven, publish-subscribe logic that characterizes the Execution Handler.

It is worth noting that the Composition Handler itself is a mashup component: any user composition action generates a Composition Handler’s event, which is notified to and managed by the Execution Handler. An interesting side effect of this architectural choice is that the logic behind the automatic component coupling is “programmable” and, therefore, flexible: it depends on a set of pre-defined listener templates configured inside the Composition Handler which, being the Composition Handler a mashup component, can in turn be easily unplugged and/or replaced.

3.3 Definition of Listeners

Dash Mash supports the definition of *default* and *custom* listeners. The former are automatically defined by the Composition Handler when a composition action is intercepted. This ensures a minimum level of inter-component synchronization that does not require users to define service coupling. Custom bindings are instead user-defined. Nevertheless, the Composition Handler offers also support by generating compatibility- and quality- based recommendations. To this aim, it dispatches the composition events to the *Recommendation Manager* that is in charge of evaluating the quality of the current composition and provides suggestions about the selection of possible components to add or substitute to the existing ones in order to achieve or improve the mashup quality.

Default Listeners. To enable the automatic definition of default listeners, we start from a classification of components. For example, in order to facilitate the construction of dashboards, it is possible to identify the following classes of components (Figure 4):

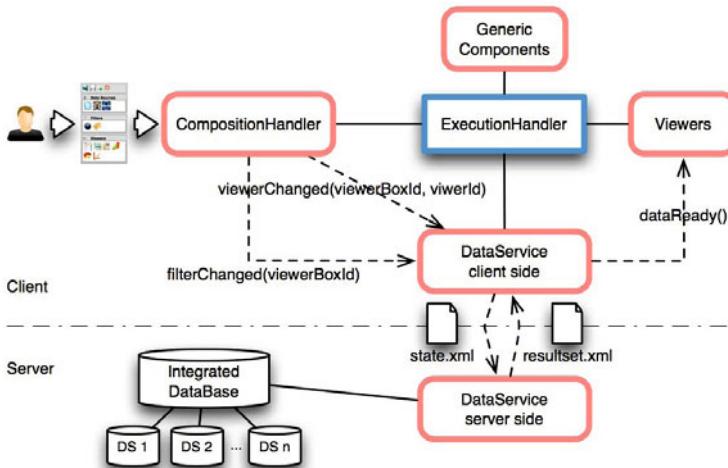


Fig. 4. Classification of components and their mutual interactions

- *Data services* are in charge of retrieving data from data sources. They are especially meant to provide access to internal (relational) data sources/warehouses. For example, in the sentiment analysis domain internal data sources store data extracted from different social applications where users post their comments. A server-side module is in charge of accessing the data sources³. A client-side module provides the actual component that synchronizes with the other mashup components to intercept changes of the composition state and send pertinent information to the server-side module so that queries for data extraction can be constructed.
- *Filters* add selection conditions over the context defined by a workspace (e.g., an interesting keyword or a time interval specified through a calendar component), thus filtering the mashup result set.
- *Viewers* support the visualization of result sets, which can be extracted by data services from internal sources or from generic external resources. Since data visualization usually involves some form of aggregation, most viewers embed a transformation logic. Therefore viewers can be simple tables, any kind of graph (as for example, those provided by Google Charts), and any visualization/aggregation service that is useful for the specific domain (e.g., a tag cloud or a map).
- *Generic components* can be also integrated to make the analysis process more effective. They can provide a variety of functionalities, such as video or image retrieval through the most common APIs, or further data sources (e.g., RSS Feeds), which can complement the information extracted from the corporate data sources.

³ When the integration of multiple sources is required, this module could embed the needed integration logics or alternatively make use of a dedicated integration layer.

The previous component classification can be adopted in several mashup contexts. For example JackBe Presto also exploits a similar classification of *mashables* [7]. What is new in our approach is that this classification of services allows us to codify default data flows that the platform exploits to automatically include ad-hoc listeners in the composition model. For example, viewers always “consume” data, as they elaborate and visualize data extracted by other components, for example data services. The addition of a viewer into the composition therefore requires at least the inclusion of a listener to subscribe the viewer to a *dataReady* event exposed by the component that produces data.

Figure 4 highlights the synchronization managed through the main default listeners. Any time a viewer is added into the workspace, the Composition Handler publishes an event that triggers a Data Service Client operation that sends a pertinent portion of the updated state to the Data Service Server. Based on the exchanged state information, the Data Service Server queries the workspace data set and sends the result set back to the Data Service Client. Based on another default listener, the Data Service Client raises an event so that subscribed viewers know about the new result set and, thus, refresh their state.

The “knowledge” about possible default mappings is coded inside the Composition Handler and can be easily configured, with the advantage that Dash Mash can be easily adapted to domains with possibly different services and service classifications. Dash Mash can also work without classification, allowing users to couple components by means of custom listeners definition.

Custom Listeners and Quality-based Recommendations. One peculiarity of Dash Mash with respect to other mashup platforms is the emphasis on quality aspects to support the users in the mashup construction. In particular, we have identified two phases where quality issues must be taken into account: (i) the registration into the platform of new services and (ii) the generation of recommendations during the mashup composition.

Registration of new services. As highlighted in Figure 2, the set of components C available in the Dash Mash platform is composed of services that can be created ad-hoc or can be public. Independently on the component nature, when a large amount of functionally equivalent data sources and services are available, quality can be one relevant driver for the selection of the most dependable components. We therefore adopt a quality evaluation approach for estimating the quality of components, which focuses on both the quality properties of mashup components and the reliability (i.e., reputation) of the Web information sources accessed to retrieve the mashup data:

- The *component quality* refers to the quality model proposed in [4], which specializes traditional quality dimensions (such as functional efficiency, reliability and usability of the APIs, and data and presentation quality criteria) to the peculiar context of mashup composition.

- The *reputation of data sources* refers to the model presented in [2], which specifically addresses the trustworthiness of Web 2.0 contents, and in particular those deriving from blogs and community sites.

When a new component is added to the DashMash component registry, the quality data needed to determine the quality indices prescribed by the two adopted quality models are documented as annotations to the component descriptors. Each component $c_i \in C$ is therefore associated with a *component descriptor* and a *quality data vector*. The component descriptor lists all the operations and the events, plus the technical details needed to compute quality indices. The quality data vector, $QD_i = [qd_{i1}, qd_{i2}, \dots, qd_{in}]$, contains the list of measures for the component quality CQ_i and source quality SQ_i as aggregated quality indexes for the i-th component and its associated data source, respectively.

Component selection during mashup composition. When a component is added into a mashup under construction, recommendations about further components to be added to the composition are generated. In particular, the Composition Handler dispatches composition events to the *Recommendation Manager* that, starting from the quality indexes stored in the component registry, suggests actions to achieve or improve the quality of the mashup. To do so, we adopt the approach described in [16], in which components are ranked on the basis of their *mashability* [16]. Mashability is defined as the capability of a component (i) to be combined with previously selected components and (ii) to maximize the quality of the overall mashup. Thus, it can be seen as a combination of two dimensions: component compatibility and aggregated quality.

Component compatibility estimates whether a component can be coupled with those already included in a composition and distinguishes between *syntactic compatibility*, checking the compatibility between input/output parameters exposed by the components, and *semantic compatibility*, checking whether input/output parameters and operations belong to the same or similar semantic categories, assuming that syntactic compatibility is satisfied. The compatibility index $comp_i$ provides a preliminary measure of the compatibility of a service c_i to be added to a given mashup: a value equal to zero indicates the incompatibility from a syntactic point of view while a positive value provides a measure of semantic compatibility. This index is stored in form of a matrix, where events and operations of the available components are related to each other and their compatibility is scored. The matrix is updated every time a new component is registered to the component registry. During mashup composition, the Recommendation Manager accesses this matrix to determine the list of components that can be suggested to the user as they can be correctly coupled with the components already in the composition. For example, in Figure 1c, only the compatible components are shown in the drop-down list where the user can select a new component to add.

While compatibility ensures the construction of “correct” mashups, *aggregated quality* drives the user in the choice of components that can lead to quality mashups. It estimates the quality of the mashup under construction as a composition of the quality of individual components. The mashup quality QM_k cannot

be however quantified as a simple sum of the quality of individual components, CQ_i , but it is necessary to weigh quality indices by taking into account the role and the importance of each component [5]. Therefore, starting from the composition model, we take into account the in- and out-degree of each component in the composition graph, and use them to determine the component impact on the overall composition, so weighing the aggregated quality. As shown in Figure 1c, aggregated quality values are visualized for each compatible components suggested in the drop-down list.

4 User-Based Validation

In order to validate the composition paradigm of Dash Mash with respect to end user development requirements, we conducted a user based study. We observed domain experts and naive users completing a set of tasks through our platform. Our goal was to assess how easily the users would be able to develop a composite application. The experiment specifically focused on the efficiency and intuitiveness of the composition paradigm, trying to measure such factors in terms of user performance, ease of use and user satisfaction. In particular, we expected all users to be able to complete the experimental tasks. However, we expected a greater efficiency (e.g., reduced completion task times) and a more positive attitude (in terms of perceived usefulness, acceptability and confidence with the tool) by expert users. Their domain knowledge and background could indeed facilitate the comprehension of the experimental tasks, and improve the perception of the control over the composition method, and thus, their general satisfaction.

The study involved 35 participants. Six of them were real end users, i.e., analysts and decision makers that are supposed to actually use Dash Mash for their analyses in the sentiment analysis domain. In order to prove to which extent the tool was intuitive even for naive users, we also involved undergrad students of the Computer Engineering Programme at Politecnico di Milano, with a moderate knowledge about Web technologies, but never exposed neither to our tool nor to the sentiment analysis domain.

For novice users, the completion of the experimental tasks was preceded by a 5-minute explanation about the domain and about the basic composition actions supported by the tool. Expert users were instead introduced to the set of available components and the basic composition mechanisms. All users were first asked to fill in a pre-test questionnaire, to gather data on their knowledge about services and mashups. All users were then asked to perform two composition tasks. The two tasks were comparable in terms of number of components to be integrated and composition steps. Task 2, however, required a less trivial definition of filters, to sift the involved data sources, and a more articulated definition of bindings. Also, while the formulation of task 1 was more procedural, i.e., it explicitly illustrated the required steps, task 2 just described the final results to be achieved, without revealing any details about the procedure required.

After the completion of the two experimental tasks, users were then asked to fill in a satisfaction questionnaire.

4.1 Results

All the participants were able to complete both tasks without particular difficulties. No differences in task completion time were found between experts and novices. In particular, domain expertise was not discriminant for task 1 ($p = .085$) and for task 2 ($p = .165$). Similarly, technology expertise was not discriminant for task 1 ($p = .161$) and for task 2 ($p = .156$). The lack of significant differences between the two groups does not necessarily mean that expert users performed bad. However, it indicates that the tool enables even inexperienced users to complete a task in a limited time. The average time to complete task 1 was about 2.5 minutes, while for task 2 it was less than 2 minutes.

The difference in completion times for the two tasks can be also used as a measure of learning [9]. This difference is about half a minute ($t = 28.2, p = .017$), i.e., a reduction of about 15%. This result highlights the learnability of the tool: although the second task was more critical compared to the first one, subjects were able to accomplish it in a shorter time.

The ease of use was also confirmed by the data collected through four questions in the post-questionnaire, asking users to judge whether they found it easy to identify and include services in the composition, to define service bindings between services, and to monitor and modify the status of the mashups. We also asked users to score the general ease of use of the tool. Users could modulate their evaluation on a 7-point scale. The reliability of the ease of use questions is satisfying ($\alpha = .75$). The correlation between the four detailed questions and the global score is also satisfying ($\rho = .58, p < .001$). This highlights the high external reliability of the measures. On average, users gave the ease of use a mark of 1.77 (the scale was from 1 very positive to 7 very negative). The distribution ranged from 1 to 4 ($mean = 1.77, meanS.E. = .12$). We did not find differences between novice and expert users. This was especially true for perceived usefulness ($p = .51$).

The user satisfaction with the composition paradigm was assessed using two complementary techniques. A semantic-differential scale required users to judge the method on 12 items. Users could modulate their evaluation on a 7-point scale (1 very positive - 7 very negative). Moreover, a question asked users to globally score the method on a 10-point scale (1 very positive - 10 very negative). The reliability of the satisfaction scale is satisfying ($\alpha = 0.76$). Therefore, a user-satisfaction index was computed as the mean value of the score across all the 12 items. The average satisfaction value is very good ($min = 1.3, max = 3.5, mean = 2.2, meanS.E. = .09$). The correlation between the average satisfaction value and the global satisfaction score is satisfying ($\rho = .41, p < .015$). On average, users gave the composition method a mark of 2.9, with a distribution ranging from 2 to 4. We did not find differences between experts and novices. Despite our initial assumption, we found that the ease of use of the tool is perceived in the same way by novice and expert users, although the latter have greater domain knowledge. The moderate correlation between the satisfaction index and the ease of use index ($\rho = .55, p = .011$) also reveals that who

perceived the method as easy also tended to evaluate it as more satisfying. This confirms that ease of use is perceived.

The last two questions asked users to judge their performance as mashup developers and to indicate the percentage of requirements they believed to have satisfied with their composition. This metric can be considered as a proxy of confidence [9]. On average, users indicated to be able to cover the 91% of requirements specified by the two experimental tasks ($min = 60\%$, $max = 100\%$, $meanS.E. = 1.7\%$). They also felt very satisfied about their performance as composers ($mean = 1.8$, $meanS.E. = .13$; 1 - very positive, 4 - very negative).

5 Related Works

Service composition has been traditionally covered by powerful standards and technologies (such as BPEL, WSCDL, etc.), which however can be mastered by IT experts only [17]. According to the End User Development vision, enabling a larger class of users to create their own applications requires the availability of intuitive abstractions and easy development tools, and a high level of assistance [3,12]. Some projects (e.g., Dyncvoker [19], SOA4All [11]) have focused on easing the creation of effective presentations on top of services, to provide a direct channel between the user and the service. However, very often such approaches do not allow the composition of multiple services into an integrated application.

Mashups are emerging as an alternative solution to service composition that can help realize the dream of a *programmable Web* [13], approachable even by non-programmer users. There is a considerable body of research on mashup tools, the so-called *mashup makers*, which provide graphical user interfaces for combining mashup services, without requiring users to write code. Among the most prominent platforms, Yahoo!Pipes (<http://pipes.yahoo.com>) focuses on data integration via RSS or Atom feeds, and offers a data-flow composition language. JackBe Presto (<http://jackbe.com>) also adopts a pipes-like approach for data mashups, and allows a portal-like aggregation of UI widgets (mashlets). IBM DAMIA [18] offers support to quickly assemble data feeds from the Internet and a variety of enterprise data sources. Mashart [6] focuses on the integration of heterogeneous components, offering a mashup design paradigm through which users create graph-based models representing the mashup composition. Marmite [20] is specifically tailored for accessing information sources: it offers a more intuitive paradigm to easily program and chain a set of operators for filtering sources. Operators provide an intelligible mechanism for extracting contents from data sources (especially from Web pages); however, the operator composition is still constrained to a parameter coupling logic based on manual type-matching.

With respect to manual programming, the previous platforms certainly alleviate the mashup composition tasks. However, to some extent they still require an understanding of the integration logic (e.g., parameter coupling). In some cases, building a complete Web application equipped with a user interface requires the adoption of additional tools or technologies. Even when they offer an easy composition paradigm, as it happens for example for Intel Mash Maker

(<http://mashmaker.intel.com>), they do not guide at all the composition process, as we do in DashMash through quality-based recommendations. A study about users' expectations and usability problems of a composition environment for the the ServFace tool [14] shows that there is evidence of a fundamental issue concerning conceptual understanding of service composition (i.e., end users do not think about connecting services). Also, in given contexts, the openness towards any kind of services and the full set of functions these tools are able to provide are not actually required. *Sandbox* environments, like the one presented in this paper, where ready-to-use services are composed according to a "constrained" integration logic, can help achieve a lightweight, easy to use composition [15], which is especially effective to serve well-defined situational purpose [10].

6 Conclusions

This paper has presented a mashup platform that leverages a composition paradigm aiding end-user development. Our platform is particularly effective for enterprise mashups, where the application domain and the component characterization are easier to identify and represent in form of rules for the automatic definition of service coupling. However, the approach remains valid for the composition of *generic* mashups. Of course, the lower the availability of domain descriptions, the greater the need for users to explicitly define service couplings. Nevertheless, as confirmed by the user-based study, the coupling definition mechanism is still intuitive and further facilitated by quality-based recommendations.

As future work, we aim at exploring different composition solutions, to address, for example, the cooperative definition of mashups (a feature that can greatly enhance team-based cooperation), also enabling mashup creation and fruition on mobile device, as well as an extension of the recommendations mechanisms based on the emergence of composition patterns from the community's mashups [16]. We also aim at easing the creation by users of DashMash components. First preliminary results concerning described services (i.e., WSDL and Linked Data) are encouraging. Our future will be also devoted to improving this feature.

References

1. Barbagallo, D., Cappiello, C., Francalanci, C., Matera, M.: A reputation-based dss: the interest approach. In: Proceedings of ENTER 2010 (2010)
2. Barbagallo, D., Cappiello, C., Francalanci, C., Matera, M.: Reputation-based selection of information sources. In: Proceedings of ICEIS 2010 (2010)
3. Burnett, M.M., Cook, C.R., Rothermel, G.: End-user software engineering. Commun. ACM 47(9), 53–58 (2004)
4. Cappiello, C., Daniel, F., Matera, M.: A quality model for mashup components. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 236–250. Springer, Heidelberg (2009)

5. Cappiello, C., Daniel, F., Matera, M., Pautasso, C.: Information quality in mashups. *IEEE Internet Computing* 14(4), 14–22 (2010)
6. Daniel, F., Casati, F., Benatallah, B., Shan, M.-C.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) *ER 2009. LNCS*, vol. 5829, pp. 428–443. Springer, Heidelberg (2009)
7. Derechin, L., Perry, R.: *Presto Enterprise Mashup Platform*. Technical report, JackBe (2010)
8. Fischer, G.: End-User Development and Meta-design: Foundations for Cultures of Participation. In: Pipek, V., Rosson, M.B., de Ruyter, B., Wulf, V. (eds.) *IS-EUD 2009. LNCS*, vol. 5435, pp. 3–14. Springer, Heidelberg (2009)
9. Hornbk, K.: Current practice in measuring usability: Challenges to usability studies and research. *Int. Journal of Human-Computer Studies* 64(2), 79–102 (2006)
10. Jhingran, A.: Enterprise information mashups: Integrating information, simply. In: *VLDB*, pp. 3–4 (2006)
11. Krummenacher, R., Norton, B., Simperl, E.P.B., Pedrinaci, C.: Soa4all: Enabling web-scale service economies. In: *Proceedings of ICSC 2009*, pp. 535–542. IEEE Computer Society, Los Alamitos (2009)
12. Liu, X., Huang, G., Mei, H.: Towards end user service composition. In: *COMPSAC*, vol. (1), pp. 676–678. IEEE Computer Society, Los Alamitos (2007)
13. Maximilien, E.M., Wilkinson, H., Desai, N., Tai, S.: A domain-specific language for web aPIs and services mashups. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) *ICSOB 2007. LNCS*, vol. 4749, pp. 13–26. Springer, Heidelberg (2007)
14. Namoun, A., Nestler, T., De Angeli, A.: Conceptual and usability issues in the composable web of software services. In: Daniel, F., Facca, F.M. (eds.) *ICWE 2010. LNCS*, vol. 6385, pp. 396–407. Springer, Heidelberg (2010)
15. Ogrinz, M.: *Mashup Patterns: Designs and Examples for the Modern Enterprise*. AddisonWesley, Reading (2009)
16. Picozzi, M., Rodolfi, M., Cappiello, C., Matera, M.: Quality-based recommendations for mashup composition. In: Daniel, F., Facca, F.M. (eds.) *ICWE 2010. LNCS*, vol. 6385, pp. 360–371. Springer, Heidelberg (2010)
17. Ro, A., Xia, L.S.-Y., Paik, H.-Y., Chon, C.H.: Bill organiser portal: A case study on end-user composition. In: Hartmann, S., Zhou, X., Kirchberg, M. (eds.) *WISE 2008. LNCS*, vol. 5176, pp. 152–161. Springer, Heidelberg (2008)
18. Simmen, D.E., Altinel, M., Markl, V., Padmanabhan, S., Singh, A.: Damia: data mashups for intranet applications. In: Wang, J.T.-L. (ed.) *Proceedings of SIGMOD 2008*, pp. 1171–1182. ACM, New York (2008)
19. Spillner, J., Feldmann, M., Braun, I., Springer, T., Schill, A.: Ad-hoc usage of web services with dynvoker. In: Mähönen, P., Pohl, K., Priol, T. (eds.) *ServiceWave 2008. LNCS*, vol. 5377, pp. 208–219. Springer, Heidelberg (2008)
20. Wong, J., Hong, J.I.: Making mashups with marmite: towards end-user programming for the web. In: *Proceedings of CHI 2007*, pp. 1435–1444 (2007)
21. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M.: A framework for rapid integration of presentation components. In: *Proceedings of WWW 2007*, pp. 923–932 (2007)

Learning Semantic Relationships between Entities in Twitter

Ilknur Celik, Fabian Abel, and Geert-Jan Houben

Web Information Systems, Delft University of Technology

Mekelweg 4, 2628 Delft, The Netherlands

{i.celik,f.abel,g.j.p.m.houben}@tudelft.nl

Abstract. In this paper, we investigate whether semantic relationships between entities can be learnt from analyzing microblog posts published on Twitter. We identify semantic links between persons, products, events and other entities. We develop a relation discovery framework that allows for the detection of typed relations that moreover may have temporal dynamics. Based on a large Twitter dataset, we evaluate different strategies and show that co-occurrence based strategies allow for high precision and perform particularly well for relations between persons and events achieving precisions of more than 80%. We further analyze the performance in learning relationships that are valid only for a certain time period and reveal that for those types of relationships Twitter is a suitable source as it allows for discovering trending topics with higher accuracy and with lower delay in time than traditional news media.

Keywords: semantic enrichment, relation learning, twitter, social web.

1 Introduction

Twitter has become an increasingly popular social network tool, not only for socializing with friends [6], but also for gathering information about what is going on in the world in whatever subject one might be interested in [8,17,7]. People typically post short status messages, interesting information they read from daily news, updates on current hot topics, their views and opinions on a certain subject matter, developments in their profession, materials about their interests, et cetera [16,12,23]. It is a growing real-time information network with several hundreds of millions of users and millions of tweets per day¹. It is estimated that highly active users regularly receive around one thousand tweets every day [4]. This information overload may cause users to get lost in the information network and become de-motivated. Finding your way around Twitter is indeed not very straightforward due to the lack of a user-friendly browsing option that would allow for a more faceted browsing experience than the existing chronologically-ordered clutter option [4,19]. Accessing required or interested fresh content easily is vital in today's information age. Hence, there is a need for guidance from users'

¹ <http://techcrunch.com/2010/06/08/twitter-190-million-users/>

point of view that would assist them in finding the related information about a subject they have just read, or where they can browse the information sources by related people, countries, cities, events, selected categories and so on.

Today, browsing and exploring micro-blogging content along certain topics is not easily supported by Twitter and other micro-blogging services. On the other hand, news websites such as Google News² provide faceted search interfaces to facilitate exploration of news. Lately, services such as the European Media Monitor³ start to exploit entities (e.g. persons, locations) for faceted search as well as relations between entities to provide recommendations. For example, if users click on a person they retrieve news related to this person and can see a list of related people. Twitter does not provide such advanced browsing functionality and the feasibility of identifying entities and relationships between entities in Twitter has not been studied yet.

Our research closes this gap and focuses on finding relationships in and between tweets based on entities in order to provide a medium where users can easily navigate through the information network by facets or easily see and access relevant content for what they are interested. For this reason, we need to understand the semantics of tweets in order to infer recommendations and support user navigation. However, Twitter posts are limited to 140 characters only. Identifying entities, and moreover relating entities, via Twitter messages is thus a non-trivial problem. In order to facilitate the identification of entities within tweets, we enrich Twitter messages by mapping them to related news articles [2]. We then apply different strategies to learn relationships between entities within tweets as well as news. Our main focus is directed at determining whether relations can be discovered from Twitter messages or whether some enrichment is needed, rather than finding the optimal strategy for relation learning. Furthermore, we analyze the validity of the discovered relations over time in order to distinguish which entities are related at a given time, and thus provide the most interesting and relevant content at the appropriate time. Our work, hence, involves engineering of a relation discovery framework which can be utilized for building useful applications such as browsing or recommendation support. The main contributions of our work can be summarized as follows:

- We implement a framework for discovering relationships between entities based on Twitter. Our framework extracts typed entities from enriched tweets/news and provides strategies for detecting semantic (trending) relationships between entities. Learnt relations provide a basis for improving browsing and content exploration as well as content recommendations in Twitter.
- Given a large dataset of more than 10 million tweets and 70,000 news articles⁴, we analyze and evaluate the framework and the different strategies for detecting relationships. In particular, we:

² <http://news.google.com/>

³ <http://emm.newsexplorer.eu/>

⁴ We make our datasets publicly available via our supporting website [1]

- investigate the precision and recall of the relation detection strategies,
- analyze how the strategies perform for each type of relationships and
- evaluate the quality and speed for discovering trending relationships that possibly have a limited temporal validity.

The paper is structured as follows. In the next section we will discuss related research. In Section 3 we will explain the design dimensions of our Twitter-based relation discovery framework and discuss different applications of the framework (e.g. browsing support). A detailed analysis and evaluation of the framework is presented in Section 4, before we discuss our main findings in Section 5 and conclude in Section 6.

2 Related Work

Twitter is a new social network phenomenon that is attracting interest from different types of people all around the world for different purposes, e.g. politicians, celebrities, journalists, scholars and the like. Over the last few years, Twitter has shown an exponential growth and became the most popular micro-blogging site. In line with this, there has been a growing interest in analyzing Twitter and its effects from a variety of research communities.

A considerable amount of research on micro-blogging and especially on Twitter focused on examining the user behavior and motivation for using Twitter [13,6], as well as user intentions [9]. Some questioned the usability of Twitter especially for novice users [19], studied capturing comprehensive locality information to be used for viewing tweets in the current local position of the mobile device [14], researched discovering and ranking fresh web sites using tweets [8], investigated the social aspects of group polarization over time in Twitter [22], while others exploited Twitter adoption and use in mass convergence and emergency events [12]. Research on temporal dynamics and information propagation on Twitter mostly focused on hashtags or observations of other sets of words in the tweets [16,17,11,20,3]. We scrutinize validity of relations between entities for given time periods and perform a temporal analysis of these relations.

Chen et al. [7] studied content recommendation in Twitter and found out that both topic and social voting (or relevance) are important considerations. They also observed that URLs extracted from the user's close social group is more successful than the most popular ones. Bernstein et al. developed *Eddi*, a prototype of an interactive topic-based browsing interface for Twitter, after observing how the users manage the incoming flood of updates in Twitter [4]. Their prototype interface groups the tweets of a user into topics mentioned explicitly or implicitly, so that users can browse for tweets of interest by topic, filtering out undesired topics. In summary, they simply categorize the tweets in the feed of the user without using any semantics or natural language processing. This approach, however, does not find the relations between the topics or perform any recommendation of related topics. While this provides a means for browsing through a user's own feed by topics, our ambition is to infer relations between

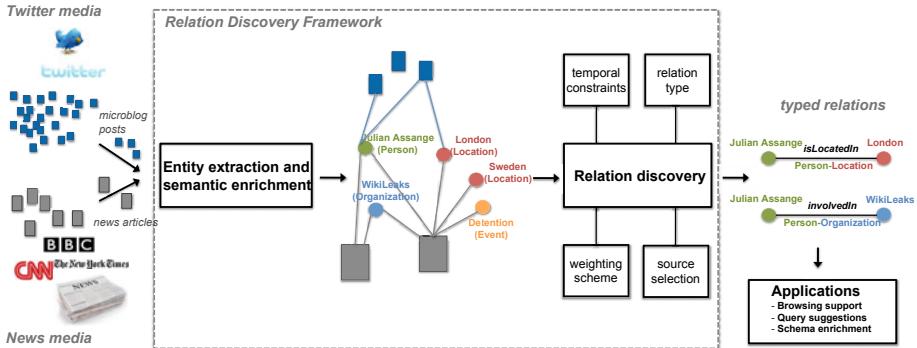


Fig. 1. Conceptual architecture of the Twitter-based relation discovery framework

entities in all tweets in order to present a list that contains the related entities of the tweet of interest even outside of the user's feed.

Tweets are limited to 140 characters and are therewith too short to extract meaningful semantics from them on their own. Furthermore, due to this limited space, users tend to use abbreviations and short-form for words in order to save space. This can only make things harder for inferring semantics from tweets. Rowe et al. [21] mapped tweets to conference talks and exploited metadata of the corresponding research papers to enrich the semantics of tweets to better understand the semantics of the tweets published in conferences. Our approach is somewhat similar to this, except we try to enrich the tweets not in a restricted domain like scientific conference, but in general. For this reason we try to map the Twitter posts we crawled with news articles on the Web over the same time period, and learn relations between them. Marinho et al. [18] proposed a method for collaboratory learning which takes a folksonomy and domain-expert ontology as input and performs semantic mapping to generate an enriched folksonomy. An algorithm based on frequent itemsets is then applied to learn an ontology over this enriched folksonomy. A similar approach is presented by Hotho et al. [10] who exploit frequent itemsets to learn association rules from tagging activities. We primarily exploit co-occurrence frequencies of entity pairs and investigate whether Twitter messages in combination with news articles can be exploited to learn relations between these entities. Learning such relationships and structures in Twitter is important to countervail information overload in Twitter and support browsing through Twitter information streams.

3 Framework for Learning Relations between Entities in Twitter

Figure 1 visualizes our approach to discovering relationships between entities. Given posts available in Twitter or the news media, our framework processes these posts and performs two main steps: (1) entity extraction and semantic enrichment and (2) relation discovery.

The first step is described in [2] and allows for the detection of entities and further semantics such as the entity type or the topic a post refers to. This process results in a graph, which connects the semantically enriched resources (tweets and news articles) with entities that are mentioned in the corresponding resources. This graph can be represented via a triple $\mathbb{G} := (R, E, Y)$, where R and E are the set of resources and entities respectively and $Y \subseteq E \times R$ is a relation between entities and resources, i.e. $(e, r) \in Y$ if entity e is referenced from resource r .

In the second step, the relation discovery strategies exploit this graph to detect pairs of entities that have a certain type of relationship (*relation type*) in a specific period of time (*temporal constraints*). Such relationships can be defined as tuples:

Definition 1 (Relationship). *Given two entities e_1 and e_2 , a relationship between these entities is described via a tuple $\text{rel}(e_1, e_2, \text{type}, t_{\text{start}}, t_{\text{end}}, w)$, where type labels the relationship, t_{start} and t_{end} specify the temporal validity of the relationship and $w \in [0..1]$ is a weighting score that allows for specifying the strength of the relationship.*

The higher the weighting score w the stronger the relationship between e_1 and e_2 . If two entities are not related then the weight is 0. Given the type of relationships that should be learnt and (possibly) temporal constraints (t_{start} and t_{end}) that prescribe for what time period the strategy should seek for such relationships, there exist two main design dimensions that influence the relation discovery (see Fig. 1).

Source selection. The source selection strategy decides which parts of the available data sources will be exploited when deducing relationships between entities, i.e. it specifies the sub-graph spanned by \mathbb{G} , which connects entities and posts (see Fig. 1), that will be exploited to discover relationships. In this paper, we differentiate between three different sources: (i) Twitter messages (\mathbb{G}_T), (ii) news articles (\mathbb{G}_N) or (iii) a combination of both media (\mathbb{G}_C).

Weighting scheme. A core challenge of the relation discovery is to compute the weight w , which expresses the strength of a relationship (see Definition 1). Those pairs of entities that are—according to the given type of the relationship—strongly related should be weighted high while for rather unrelated entities the weight should be low. In this paper, we utilize the co-occurrence frequency of two entities as weighting scheme. Hence, given a certain data source, we count the number of resources (tweets and/or news articles), in which both entities e_1 and e_2 are mentioned: $w_{co}(e_1, e_2) = |r \in R | (e_1, r) \in Y \wedge (e_2, r) \in Y|$.

As depicted in Fig. 1, our Twitter-based relation discovery framework outputs semantic relationships between entities that support various applications. For example, relationships between entities can, in combination with the enriched data \mathbb{G} , be applied to enable browsing support by linking to related tweets. Furthermore, when people search in Twitter, our framework can identify entities

mentioned in a keyword query and exploit relationships to other entities to provide query suggestions to the user. In addition to these applications, our relation discovery framework can assist in building ontologies on the fly. As our framework provides functionality to map entities to concepts of existing ontologies such as DBpedia [5], learnt relationships can also be applied to complement and enrich existing knowledge bases and schemata.

We implemented the relation discovery framework as a Java library using the Twitter streaming API⁵ to monitor tweets, boilerpipe⁶ [15] for extracting the main content of news articles from websites such as BBC or CNN and OpenCalais⁷ for extracting entities from tweets and news articles respectively.

4 Evaluation of Strategies for Learning Relationships

Given the relation discovery framework introduced in the previous section, we evaluate the performance of the different strategies and investigate the following research questions.

1. Which media source (Twitter, News or a combination of both) allows for the highest precision and recall in detecting relationships between entities?
2. Does the accuracy depend on the type of entities which are involved in a relation that should be discovered, and, if so, for what type of relations is the accuracy high/low?
3. How do the strategies perform for discovering relationships which have temporal constraints, and how fast can the strategies detect (trending) relationships?

4.1 Data Collection and Dataset Characteristics

To answer the above research questions, we applied the relation discovery framework and monitored Twitter as well as news media over a period of two months starting from 15 November 2010. Overall, we crawled more than 10 million tweets and more than 70,000 news articles, which were published by prominent news Web sites such as BBC, CNN and New York Times. Figure 2(a) shows the number of Twitter posts and news articles published during the observation period. For news, the number of news articles published per day ranges between 100 and 1000 articles and follows rather characteristic patterns: on weekends the number of articles decreases clearly. Regarding Twitter posts, we see that the amount of tweets published each day is much higher and varies much stronger (between 50,000 and 400,000). However, we could not detect correlations between weekdays/weekends and the number of Twitter messages posted on those days. Two of the minima were caused by temporary unavailability of the Twitter monitoring service.

⁵ http://dev.twitter.com/pages/streaming_api

⁶ <http://code.google.com/p/boilerpipe/>

⁷ <http://www.opencalais.com/>

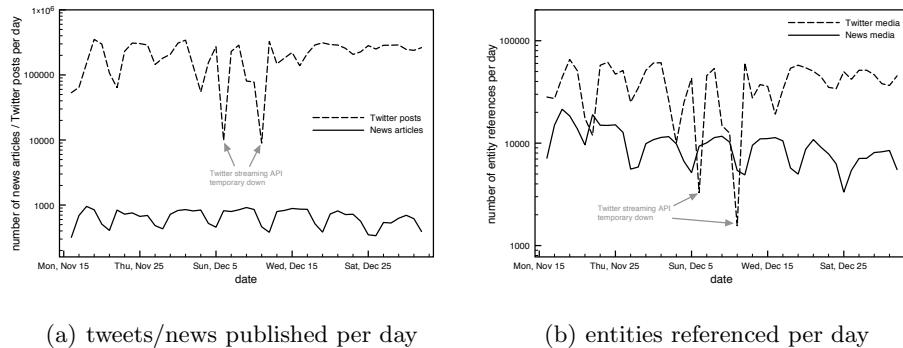


Fig. 2. Dataset characteristics: (a) the number of Twitter posts and news articles published per day and (b) the number of entity references identified per day

We processed all Twitter messages and news articles via our framework to extract entities that are mentioned in the tweets and news respectively. The number of entity references per day that we obtained by this extraction process is depicted in Figure 2(b). It is interesting to see that the amount of entity references observed in Twitter (approximately 10,000-100,000 references per day) is higher than for the News (approximately 5,000-20,000 references per day), but does not differ as strongly as for the number tweets/articles published each day. In fact, it is much harder to detect entities in short Twitter messages than news articles. For more than 40% of the processed tweets, we could not detect any entity while for 99.3% of the news articles we succeeded in detecting at least one entity. Hence, news articles seem to be a more valuable source for detecting entities than tweets. This observation gives our approach of exploiting news for detecting relations among entities mentioned in Twitter the first justification.

However, do the entities mentioned in Twitter overlap with those mentioned in news articles? To answer this question, we measured the overlap of entities that occur in both the Twitter media and the news media. For the given period, we observe that 72.6% of the top 1000 mentioned entities in Twitter are mentioned in the news media as well. This observation further validates the feasibility of considering news articles to detect relations between entities mentioned in Twitter and vice versa.

Figure 3 lists the number of distinct entities we observed in Twitter for the 39 different entity types. Persons, locations and organizations were mentioned most often, followed by movies, music albums, sport events and political events. In our analysis, we investigate the discovery of relationships between these entities. Moreover, we analyze specific types of relations such as relationships between persons and locations or organizations and events in detail. The complete list of those relation types that we analyzed in detail can be obtained via the supporting website [1].

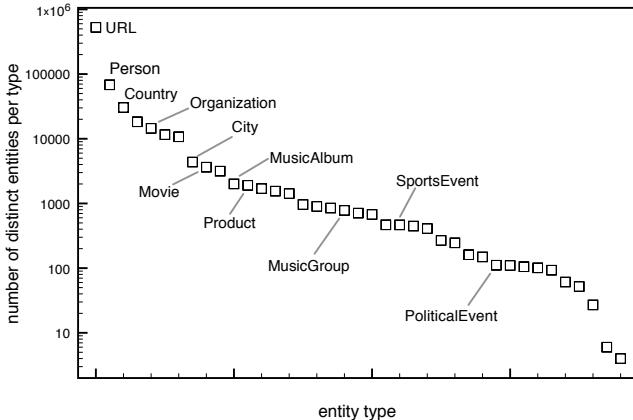


Fig. 3. Number of distinct entities per entity type mentioned in Twitter (39 different types)

4.2 Methodology, Metrics and Ground Truth

The relation detection task can be interpreted as a ranking problem. Given the dataset described above, the relation detection strategies have to rank entity pairs (of given types) so that those pairs, which are *truly* related (at a specific point in time) appear at the top of the ranking. We obtained the corresponding ground truth of *true* relationships (i) via DBpedia [5] and (ii) via a user study.

To utilize DBpedia—the structured representation of Wikipedia—as ground truth, we mapped entities to their corresponding DBpedia resources by exploiting the DBpedia lookup service⁸. We considered a given entity pair (e_1, e_2) as related if e_1 referenced or mentioned e_2 in one of its properties (e.g. in the full text description) or vice versa. However, for more than 35% of the entities we could not retrieve appropriate DBpedia URIs, for example, because these entities were not mentioned in DBpedia (yet). Therefore, we also conducted a user study to complement the DBpedia-based ground truth.

In our user study, we asked PhD students from our group to judge whether two entities, for which a relationship was detected by one of the tested strategies, are really related on a four-point scale: “not related or unknown”, “rather not related”, “rather related” and “related”. Overall, we obtained 1294 judgments. 62.6% of the relationships were rated as “related” and were therefore considered as ground truth. We further asked the participants to judge whether a relationship is valid at a specific period in time. For example, whether they think that “Julian Assange (person) and London (city)” or “Bill Clinton (person) and presidential election (political event)” are related during a certain time-frame (one week). Given such temporal constraints, 57.3% of the relations were classified as related.

Given the ground truth, we measured the quality of the relation discovery strategies by means of the following information retrieval metrics.

⁸ <http://dbpedia.org/lookup>

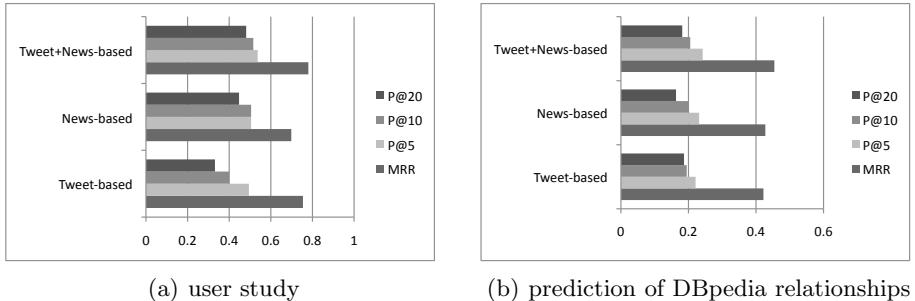


Fig. 4. Accuracy of different relation discovery strategies based on (a) ground truth obtained from the user study and (b) ground truth obtained from DBpedia (more than 5,000 relations).

MRR. The *MRR* (Mean Reciprocal Rank) indicates at which rank the first *true* relationship occurs on average.

P@k. Precision at rank k (*P@k*) represents the average proportion of *true* relationships within the top k.

R@k. Recall at rank k (*R@k*) is the number of *true* relationships, which are listed within the top k, divided by the number of known *true* relationships.

F@k. F-measure at rank k (*F@k*) is the harmonic mean of R@k and P@k.

4.3 Results

To answer the first research question, we summarize the accuracies of the different strategies in Figure 4. For both test settings – user study and prediction of relationships in DBpedia – we see that by combining the Tweet-based and News-based strategies (Tweet+News-based), we achieve higher precision. In particular, the Tweet+News-based strategy performs better with respect to the mean reciprocal rank (MRR), P@5 and P@10. The results from the user study (Figure 4(a)) reveal that the Tweet-based strategy performs slightly better than the news-based strategy regarding MRR. However, the precision of the Tweet-based strategy decreases stronger than the precision of the News-based strategy when *k* is increased. In fact, this observation is influenced by the number of candidate relationships: all strategies are based on the co-occurrence of entities and, as tweets are limited to 140 characters in length, we observe that the probability of co-occurring entities is smaller in Twitter. For example, for relationships between *persons* and *political events* the probability that such entities co-occur in a tweet of our Twitter dataset is 0.00025, which results in 146 co-occurring persons-event candidate pairs. In contrast, the news corpus features a probability of 0.32 and 7893 person-event co-occurrences in total. Hence, the consideration of news promises to detect further relationships and therewith can potentially increase recall.

Figure 5 confirms this hypothesis. With respect to recall (R@10) and F-measure, the News-based strategy performs (32%) significantly better than the Tweet-based strategy (see Figure 5(a)). By exploiting both tweets and news, we

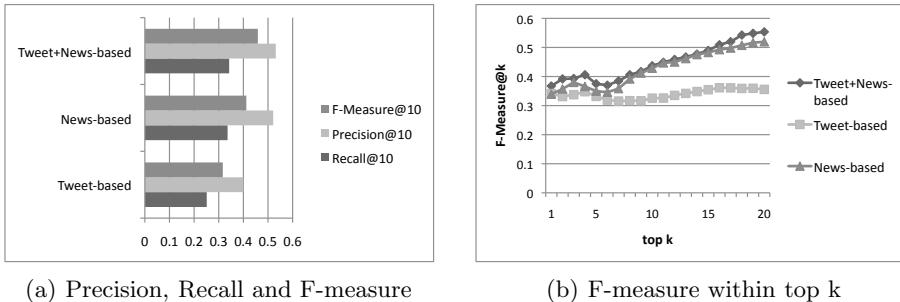


Fig. 5. Precision and recall: (a) precision, recall and F-measure within the top 10 and (b) F-measure for different ranks

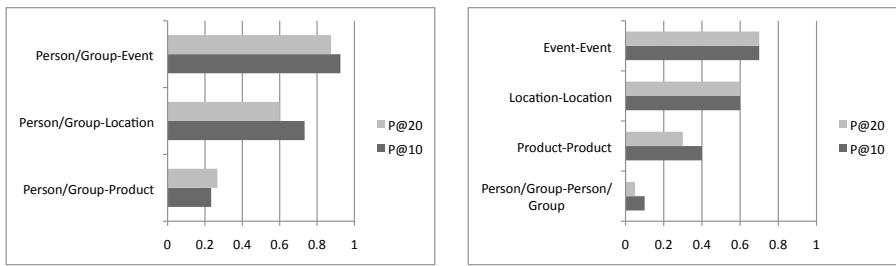
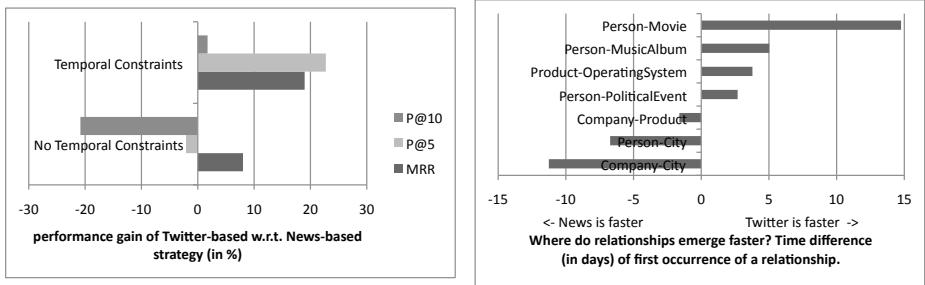


Fig. 6. Accuracy of different types of relations: (a) relations of persons or groups with events, locations and products and (b) relations between entities that are of the same type.

achieve even higher performances regarding recall (with 36%) and F-measure (with 45%). Figure 5(b) further illustrates how the F-measure performs at different ranking levels. While the F-measure of the Tweet-based strategy saturates quickly, the F-measure of the Tweet+News-based strategy increases and continuously achieves higher F-measure than the other strategies. We conclude that by combining both tweets and news articles, we achieve the best performance (regarding recall, precision and F-measure) in discovering typed relationships among entities – which answers the first research question raised at the beginning of this section.

Figure 6 depicts the accuracy of the Tweet+News-based strategy for specific types of relations and allows us to answer the second research question. To better visualize the results, we grouped the different types of relations into more abstract classes. For example, *Person/Group-Event* relationships cover relations between persons and political events, persons and sport events, organizations and sport events, et cetera. Figure 6(a) shows the precisions for inferring relationships between persons or groups and (i) events, (ii) locations (including cities or countries) and (iii) products (including movies or music albums). It is



(a) Relations with(out) temporal constraints. (b) Time difference in detecting relations.

Fig. 7. Temporal aspects: (a) difference in accuracy between news-based and tweet-based strategies for relations with/without temporal constraints and (b) time difference between news-based and tweet-based strategies in detecting certain relations

interesting to see that the Tweet+News-based strategy discovers relationships between persons/groups and events with higher precision—0.92 and 0.87 regarding P@10 and P@20—than people’s relations to products (0.23 and 0.26) or locations (0.73 and 0.6). The good performance in the context of events can be explained by the nature of tweets and news articles, which are exploited to detect the relationships. Both media are centered around news events [16] which makes them a good source for inferring relationships between events and other entities.

Figure 6(b) shows that relationships between two events can also be discovered with high precision, followed by relations between locations (e.g. countries and cities, natural features and countries). For relationships among products (e.g. mobile devices and operating systems), the Tweet+News-based detection strategy performs better than for relationships between people, for which we observe the lowest performance (0.1 for P@10). However, in the context of relations between people, Twitter seems to be a much more reliable source. The Tweet-based strategy achieves, for example, a precision of 0.4 (P@10) for inferring relationships between persons and music groups while the News-based strategy allows for only 0.1 (P@10). One reason for this performance difference is the number of distinct entities that co-occur in a news article or Twitter message. On average, each news article contains 19.3 entities while a Twitter message allows for the identification of, on average, less than 1.8 entities. Intuitively, one can thus interpret these observations as follows: if two entities co-occur in a tweet then there is a high chance that there exists also a relationship between these entities (e.g. in DBpedia).

Some relationships have temporal constraints. Relations between persons and organizations or persons and locations might be valid or trending only for a certain period in time. For example, in December 2010, *London* and *Julian Assange*, the founder of WikiLeaks, were highly related, because Assange was under arrest in London at that time. The identification of such trending relationships could improve topic-based Twitter and news exploration interfaces. Figure 7(a) illustrates that Twitter is more appropriate for inferring relationships, which have

temporal constraints, than the news media. The Tweet-based strategy improves precision ($P@5$) by 22.7% in comparison with the News-based strategy, which in turn performs slightly better—with respect to $P@5$ and $P@10$ —for detecting relationships without temporal constraints.

For the identification of trending relationships, it is further interesting to know which source of information allows for fast discovery of relationships. Figure 7(b) shows that particularly relationships between persons and movies or music albums emerge much faster (14.7 and 5.1 days respectively) in Twitter than in the traditional news media.

5 Discussion

Our results prove the feasibility of learning relationships between entities based on Twitter messages. However, we observed that the Tweet-based strategy achieves poorer results, with respect to the precision within the top k , when we increase the k value, while more stable behavior was noticed by the News-based strategy under the same conditions. On the other hand, the high MRR value for the Tweet-based strategy suggests that if two entities do co-occur in such a limited-size space, then there is a higher probability of strong correlation between them. This statement, however, does not apply to news articles where there are more identified entities on average than for tweets. Therefore, a strategy that combines the news and tweets to increase the probability of identifying entities, and then looking at their co-occurrence frequency to detect more relations, namely the Twitter+News-based strategy, takes the best out of the two in order to produce the highest performance (recall/precision) for discovering relations.

We described already one application of this relationship discovery in our motivations: content exploration in Twitter. When a user is reading a tweet, we can provide a list of entities that are related to the entities in the current tweet with the aid of our learnt relations. This allows users to follow a link in a chosen category (entity type) that they want to explore more, facilitating faceted browsing. For example, given a single Twitter message that mentions a person, learnt relationships link events, organizations or products related to this person and can thus be used to explore related content.

Having identified the best strategy and the media source for detecting relations between entities, we further investigated whether the accuracy varied depending on the entity and relation types. We observed that our Twitter+News-based strategy performs well in detecting relations between events and other types of entities, but not so well for relationships among persons. We see one possible explanation in the disambiguities in some of the typed entities, especially with synonymous entity values. For example, we observed cases where the same person was identified in different ways in the Person entity, such as “Barack Obama”, “Obama”, “Mr. Obama” and “President Obama”. The co-occurrence-based strategies treat each of these representations as individual entities without merging them.

On the other hand, we can see from the results that relations among events and among locations can be discovered with high success. Such relations can be used to suggest further sources of information to a user who is reading up on an event in order to point them in the direction of related events. For example, a user viewing a tweet about the “Grammy Awards” to be held in Los Angeles on the 13th February might be interested to read about the “Academy Awards”, which will happen later that month in the same city. Providing trending relations as a separate “list of suggestions” to the user would also enhance browsing means and the ability to access (interesting/desired) information when the information is still relevant. For this reason, we examined the quality and speed for discovering relationships with temporal dynamics in order to evaluate our strategies with media sources to eventually find trending relations. Tweet-based strategy was observed to perform highly in detecting relations with temporal validity both in terms of precision and speed when compared with the News-based strategy.

Our analysis of how fast we can discover relations further showed that the source of information first comes out in tweets, before it appears in the news especially for certain types of relations, e.g. Person-Movie, Person-MusicAlbum, Product-OperatingSystem and Person-PoliticalEvent. A possible explanation for Twitter’s considerable speed and accuracy in finding “trending topics” over news media could be explained by comparing the number and frequency of tweets to that of the news articles posted, as was shown in Figure 2(a). Another explanation for this could be the fact that Twitter is used as a tool for fast communication, where news media is mostly for information broadcasting/reporting. Looking at the graph in Figure 2(a), one can see that there is more of a regular pattern for the news articles, while the number of tweets are more irregular, suggesting that they vary according to the current trend(s) more so than the day of the week. Therefore, there is no surprise that Tweet-based strategy has better performance in detecting relations with temporal validity. This also describes why relations types such as Company-Product, Person-City and Company-City are detected faster with news media than Twitter. These relations are long-term relations in general with less temporal constraints in the relation.

6 Conclusions

In this paper, we proposed a framework for discovering relationships between entities mentioned in microblog posts published on Twitter. Our framework extracts entities from Twitter messages, allows for further enrichment by exploiting traditional news media and provides different strategies for detecting typed relationships that moreover may have temporal constraints. Our analysis revealed that the consideration of news articles improves recall and precision of the relation discovery clearly. Furthermore, the accuracy depends strongly on the type of relationships that should be learnt: while relations among persons are fairly difficult to detect, relationships between persons and events can be discovered with high precision of more than 80%. For detecting relationships that have a limited temporal validity, Twitter-based strategies outperform News-based strategies by

more than 20% regarding precision (P@5). Further, by exploiting Twitter it is possible to discover certain types of trending relationships faster than by considering traditional news media.

Relations learnt by our relation discovery framework allow for supporting topic-based content exploration in Twitter and can be applied to suggest keywords when querying for Twitter content. Moreover, relationships can be exploited to enrich and complement existing ontologies, for example, by exploiting the linkage to DBpedia entities as provided by our framework. The relation discovery framework can also be used for engineering content recommendation applications. Studying the performance of these applications is part of our future work.

Acknowledgments. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no ICT 257831 (ImREAL project⁹).

References

1. Abel, F., Celik, I.: Supporting website: datasets, further details and additional findings (2011), <http://wis.ewi.tudelft.nl/icwe2011/relation-learning/>
2. Abel, F., Gao, Q., Houben, G.J., Tao, K.: Semantic Enrichment of Twitter Posts for User Profile Construction on the Social Web. In: Antoniou, et al. (eds.) Extended Semantic Web Conference (ESWC), Heraklion, Greece, Springer, Heidelberg (2011)
3. Akcora, C.G., Bayir, M.A., Demirbas, M., Ferhatosmanoglu, H.: Identifying Breakpoints in Public Opinion. In: Melville, P., Leskovec, J., Provost, F. (eds.) Proceedings of Workshop on Social Media Analytics (SOMA) at KDD 2010, Washington, DC, USA (2010)
4. Bernstein, M., Kairam, S., Suh, B., Hong, L., Chi, E.H.: A torrent of tweets: managing information overload in online social streams. In: Proceedings of the CHI Workshop on Microblogging: What and How Can We Learn From It? (2010)
5. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia - A crystallization point for the Web of Data. In: Web Semantics: Science, Services and Agents on the World Wide Web (2009)
6. Cha, M., Haddadi, H., Benevenuto, F., Gummaduri, P.K.: Measuring User Influence in Twitter: The Million Follower Fallacy. In: Cohen, W.W., Gosling, S. (eds.) Proceedings of the Fourth International Conference on Weblogs and Social Media (ICWSM). The AAAI Press, Washington, DC, USA (2010)
7. Chen, J., Nairn, R., Nelson, L., Bernstein, M., Chi, E.: Short and tweet: experiments on recommending content from information streams. In: Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI), pp. 1185–1194. ACM, New York (2010)
8. Dong, A., Zhang, R., Kolari, P., Bai, J., Diaz, F., Chang, Y., Zheng, Z., Zha, H.: Time is of the essence: improving recency ranking using twitter data. In: Proceedings of the 19th International Conference on World Wide Web (WWW), pp. 331–340. ACM, New York (2010)

⁹ <http://imreal-project.eu>

9. Honeycutt, C., Herring, S.C.: Beyond microblogging: Conversation and collaboration via twitter. In: Proceedings of the 42nd Hawaii International Conference on Systems Science (HICSS), pp. 1–10. IEEE, Big Island (2009)
10. Hotho, A., Jäschke, R., Schmitz, C., Stumme, G.: Emergent Semantics in BibSonomy. In: Hochberger, C., Liskowsky, R. (eds.) Informatik 2006: Informatik für Menschen. LNI, vol. 94(2). GI, Bonn (2006)
11. Huang, J., Thornton, K.M., Efthimiadis, E.N.: Conversational Tagging in Twitter. In: Chignell, M.H., Toms, E. (eds.) Proceedings of the 21st ACM Conference on Hypertext and Hypermedia (HT), pp. 173–178. ACM, New York (2010)
12. Hughes, A.L., Palen, L.: Twitter Adoption and Use in Mass Convergence and Emergency Events. In: Landgren, J., Jul, S. (eds.) Proceedings of the International Conference on Information Systems for Crisis Response and Management ISCRAM 2009 (May 2009)
13. Java, A., Song, X., Finin, T., Tseng, B.: Why we twitter: understanding microblogging usage and communities. In: Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis. WebKDD/SNA-KDD 2007, pp. 56–65. ACM, New York (2007)
14. Kaufman, S.J., Chen, J.: Where we Twitter. In: Proceedings of the CHI Workshop on Microblogging: What and How Can We Learn From It? (2010)
15. Kohlschütter, C., Fankhauser, P., Nejdl, W.: Boilerplate detection using shallow text features. In: Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM), pp. 441–450. ACM, New York (2010)
16. Kwak, H., Lee, C., Park, H., Moon, S.: What is twitter, a social network or a news media? In: Proceedings of the 19th International Conference on World Wide Web (WWW 2010), pp. 591–600. ACM, New York (2010)
17. Lerman, K., Ghosh, R.: Information contagion: an empirical study of spread of news on digg and twitter social networks. In: Proceedings of 4th International Conference on Weblogs and Social Media, ICWSM (2010)
18. Marinho, L.B., Buza, K., Schmidt-Thieme, L.: Folksonomy-based collabulary learning. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T.W., Thirunarayanan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 261–276. Springer, Heidelberg (2008)
19. Owens, J.W., Lenz, K., Speagle, S.: Trick or Tweet: How Usable is Twitter for First-Time Users? Usability News 11 (2009)
20. Romero, D.M., Meeder, B., Kleinberg, J.: Differences in the mechanics of information diffusion across topics: Idioms, political hashtags, and complex contagion on twitter. In: Proceedings of the 20th International Conference on World Wide Web (WWW). ACM, New York (2011)
21. Rowe, M., Stankovic, M., Laublet, P.: Mapping Tweets to Conference Talks: A Goldmine for Semantics. In: Passant, A., Breslin, J., Fernandez, S., Bojars, U. (eds.) Workshop on Social Data on the Web (SDoW), Colocated with ISWC 2010, CEUR-WS.org, Shanghai, China, vol. 664 (2010)
22. Yardi, S., boyd, d.: Dynamic Debates: An Analysis of Group Polarization over Time on Twitter. Bulletin of Science, Technology and Society 30 (2010)
23. Zhao, D., Rosson, M.B.: How and why people Twitter: the role that micro-blogging plays in informal communication at work. In: Proceedings of the ACM International Conference on Supporting Group Work (GROUP), pp. 243–252. ACM, New York (2009)

Mobile Mashup Generator System for Cooperative Applications of Different Mobile Devices

Prach Chaisatien, Korawit Prutsachainimmit, and Takehiro Tokuda

Department of Computer Science, Tokyo Institute of Technology
Meguro, Tokyo 152-8552, Japan
`{prach,korawit,tokuda}@tt.cs.titech.ac.jp`

Abstract. This paper presents a development and an evaluation of a mobile mashup generator system to compose mobile mashup applications and Tethered Web services on a mobile device (TeWS). With less programming efforts, our system and description language framework enables a rapid development, a reusability of working components and a delivery of new cooperative mobile mashup applications. Working components in the mashup execution are derived from a combination of existent mobile applications, JavaScript automated Web page data extractions, and RESTful Web service consumptions. The state of art generator system is evaluated with novice and expert composer groups, to validate the usability of the system and the expressibility of our Mobile Application Interface Description Language (MAIDL). Complex mashup examples are provided to demonstrate new cooperative applications of generated Web services, which enable platform-independent functionality exchange across devices via tethered HTTP communications.

Keywords: Mobile mashup application, description language, tethered Web service, mobile Web server.

1 Introduction

A software development for mobile devices is becoming an essential task in the field of information technology. The devices' unique capabilities such as on-the-go Internet access, GPS and camera-based applications are becoming the key functional components in developing modern mobile applications. The major drawback when creating a multi-platform mobile Web application is that it tends to make less use of the device's useful features. Mobile mashup development using the device's native programming language requires more explicit knowledge and does not allow the mashups to be developed as fast as the Web-based ones are.

As an alternative, the second iteration of approaches is related to the code-to-code (C2C) [1], [2] and model-to-code (M2C) [3] approaches. The C2C approach, however, is not designed for developers (or mashup composers) without programming skills. This approach emphasizes on a conversion of Web language to the device's native programming language and allows sensors be accessed with less

code having to be written. On the other hand, the M2C approach tends to promote Web mashup without the integration of mobile devices' unique features.

Our study addresses problems found in these platform centric and model-code centric approaches by providing a fast-paced development using an XML-based description language called Mobile Application Interface Description Language (MAIDL) and its mashup composition tool. With less programming efforts, composers can freely integrate parts of Web information (annotated Web navigations, and queries from Web services) and mobile devices' unique features (existent mobile applications accessing device's sensors). We proposed automatic code generation algorithms for mobile mashup application, which are *Mashup Output Context Transformation* and *Mashup Process Scheduling* algorithm. The output application can be designated for a single device, as a normal mobile application, and for multiple devices, as a Tethered Web service on the mobile device (TeWS). Thus, platform-independent communication between devices for functionality exchange and cooperative application can be simply created.

2 Related Work

Our previous study [4] proposed a composition model for mashup of Web applications, Web services and mobile applications. This approach shows the reusability of Web information and mobile devices' capabilities to outline a mashup application. As a continuation, this research aims at the composition tool to deliver the *End Users Mashup Development on Mobile Devices*. We also extend our approach's expressibility to compose cooperative mashups for multiple devices via a TeWS. The contrary point of our approach to the conventional mobile Web mashup [5] is that other approaches extensively create or reuse Web information as a part of user interface without integrating devices' sensors and existent application. Mashups also cannot be created as a TeWS.

In academic research, mashup approaches are often referred to as a M2C approach in which a composer's created composition model is later translated to programming code. These approaches allow the integration of Web application from many Web application components. In applying these models to mobile mashups, the runtime of mashups on actual devices is not as powerful as the one used in PCs or Web servers. In order to reuse the same composition pattern on a mobile device, the generation and execution procedures of the mashup application have to be adjusted for a limited computation environment [6].

In contrary to the M2C approach, the recent C2C mobile development approaches are designed so that developers use Web languages to build multi-platform mobile applications. Despite the C2C advantage in its Web language compatibility, its frameworks are not designed for non-programmers. This research is based on the M2C approach, which consists of 2 parts, M2M (model abstraction) and M2C (automatic software generation).

Code, which is generated from MAIDL, is in a procedural paradigm since the control part mainly consists of procedures that are passing results and synchronizing processes in the mashup runtime environment. For this reason, we proposed automatic code generation algorithms, which assist composers in creating

mobile mashup applications. Moreover, the final output is not limited to mobile application as traditional methods are [7] [8]. A TeWS can be generated and later consumed by other clients. Later in complex mashup examples, we show how the composed TeWS is applied to platform-independent communications between devices. In addition to the main composition tool, the Web extraction assistant tool (WXTractor) in this research is created based on methods of Web APIs and Web automation [9] and partial information extraction [10]. The tool automatically generates mashup execution parameters for MAIDL from an annotated tag of an HTML document. To expose these features to novice composers, our tool also visually provides result samples in the model abstraction process.

3 An Overview of Our Research Approach

3.1 Objective and Motivation

1. *Explore a mobile mashup model.* The conventional disciplines discussed in section 2 show that a mashup model for the mobile mashups is not concretely defined. We aim to find an optimal mashup model which leads to a better solution in creating mashups for mobile devices.
2. *Deliver reusability.* Our mashup components include existent mobile applications and Web information. Therefore, developing mashups with low-level API, such as creating an image recognition component with a new algorithm, is beyond our research scope.
3. *Enable fast prototyping.* Mashups can be created from a Web-based software generation tool. Composers are allowed to generate source code, compile, and test it immediately after the composition model is correctly prepared. Methods called *Mashup Output Context Transformation* and *Mashup Process Scheduling Algorithm* would assist composers by automatically managing foreground and background runtime behaviors of the mashup components.
4. *Target novice and expert composers.* Most of the abstraction models are designed for programmers. Through our composition tool, we aimed to let novice composers be able to create mashups for mobile devices. More advanced and customizable features are added for expert composers.
5. *Demonstrate a TeWS.* A mashup in our approach can be created as a mobile application to run on a device or as a TeWS. Functionality exchanges and interactive collaborations between devices can be derived from our approach, and these are unique contributions which do not appear in other approaches.

In order to run most flexible configuration on mobile devices (such as third party mobile applications and embedded server modules), we use the Android open source platform [11] as our mashup runtime environment.

3.2 MAIDL and its Abstract Model Composition

The general concept of MAIDL (shown in Fig. 1) is to provide data flows between mashup components for its execution and output. The components consist of:

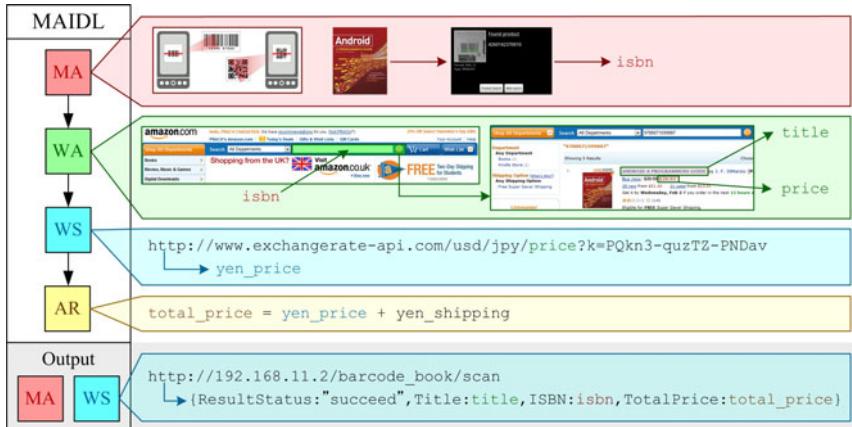


Fig. 1. Overview of MAIDL and its abstract model composition

1. Web Application Component (WA). A part of a Web page or a query through HTML forms can be reused through a WA component. Composers are provided with a tool called WXTractor to annotate HTML tags and specify execution commands. JavaScript code will be generated according to the specification and executed in the runtime environment on the mobile device.
2. Web Service Component (WS). Connections to REST Web services are applicable to our composition. Composers specify a URL and a query expression (such as XPath or JSON dot notation) to access a part of the whole data.
3. Mobile Application Component (MA). A part of mashup execution can be derived from a mobile application. Our method allows an application which implemented Intent and Service messaging protocol [12] to be integrated.
4. Arithmetic Component (AR). A mathematical operation between results from one or more components can be performed through AR. The operation includes addition, subtraction, division, multiplication, summation, comparison, and GPS distance calculation from 2 pairs of GPS coordinates.

The composition process begins where composers select the output context (mobile application or TeWS). Components can be added in any order but not starting with the AR component. Composers then configure each component's parameter according to their data flows and logical specifications. Results from the publisher components attached in the upper hierarchical order are listed and are selectable in the nested subscriber components. Finally, composers configure the output component and export the abstracted model to a MAIDL script file.

3.3 Mashup Mechanism, Output Context and Process Scheduling

The mechanism in each output context is different. Therefore, it is crucial that composers specify the context first. The mashup composition tool is

Table 1. Mashup mechanism in a mobile application and a TeWS output context

Output Context	Runtime Process	Inter-Component Messaging Protocol	Working process in detail
MA	Foreground (FG)	WA:Intent(FG) or Service(BG) WS: Service(BG)	WA: Custom browser and JavaScript navigation WS: HTTP connection and message retrieval/query MA: Intent(FG) or Service(BG)
TeWS	Background (FG)	WA:Intent(FG/SMA) or Service(BG) WS:Service(BG) MA: Intent(FG/SMA) or Service(BG)	Same as MA output context

context-sensitive and will allow integrations of only compatible components. Table 1 shows the mashup mechanism in each output context.

Mashup Mechanism in a MA Context. Components in this context work separately as mobile applications. WA components employ a custom Web browser and JavaScript navigation sequences generated from MAIDL. In the case where form submissions (e.g. fill a form with keywords and click on a submit button) or user interactions (e.g. select a link from search results) are required, the runtime will work in a foreground process. When a part of a static Web page is extracted without user interaction, the WA components work in a background process. WS components always run in a background process. The process includes HTTP connections, message retrievals and queries of the data. The runtime of MA components depends on its messaging protocol. If the Intent protocol is used, they work in a foreground process. When the Service protocol is used, they work in a background process. After all components finished their tasks, all parameters will be passed to the final output mobile application according to the configuration.

Mashup Mechanism in a TeWS Context. A mashup runtime in a TeWS output context does not allow MA components to be called directly, for the reason that the process in WS component itself runs as a background process. Therefore we use a *Switcher Mobile Application* (SMA) to indirectly call each MA component in the same way the mechanism works in the MA output context. Fig. 2 shows the messaging processes in a normal direct call and an indirect call via SMA. The final TeWS, is generated from results passed from each component. The TeWS can be accessed using the device's IP address and the path specified in the project's MAIDL script. For performance reasons, we applied the REST architecture to the TeWS output, to deliver JSON messages.

Mashup Process Scheduling. Processes in a linear FIFO order are scheduled according to the hierarchical order of a publisher-subscriber pattern and their runtime behavior. If no parallel execution occurred, a process will sequentially wait for prior processes to finish its task. In a parallel execution, processes under a

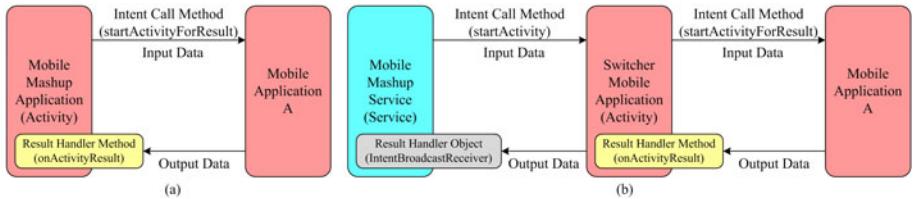


Fig. 2. Messaging processes using a direct call (a) and an indirect call (b)

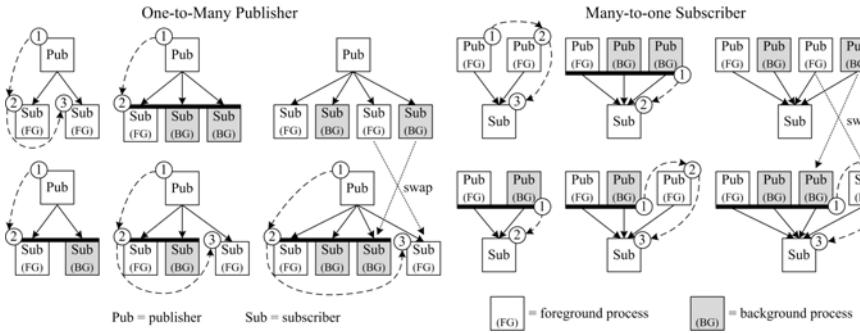


Fig. 3. *Mashup Process Scheduling Algorithm.* This figure shows data flows (solid lines), task sequences (dashed lines), merged sequences (black bars) and swaps (dotted lines)

one-to-many publisher are executed from a top-to-bottom order as defined in the MAIDL script file. Many-to-one subscribers or the terminal output component wait for all prior components to finish before its own execution begins.

The process synchronization in our runtime environment employs a Java Thread to wait for messages sent from components when its task is finished. As demonstrated in Fig. 3, the algorithm merges execution sequences of background processes to the latest foreground process in the same hierarchical order. The process order is swapped to execute background processes with the first foreground process in the same hierarchical order. The equations below shows how the total execution time t is reduced to t' after the algorithm is applied. i and j are process indexes after the algorithm is applied, where $n + m \geq i + j$

$$t = \sum_{N=0}^{n=0} t(F_n) + \sum_{M=0}^{m=0} t(B_m) . \quad (1)$$

$$t' = \begin{cases} \sum_{I=0}^{i=0} t(F_i) + \sum_{J=0}^{j=0} t(B_j) & \text{if all } t(F_n) \geq t(B_m) . \\ \sum_{M=0}^{m=0} t(B_m) & \text{if some } t(F_m) < t(B_m) . \\ \sum_{N=0}^{n=0} t(F_n) & \text{if all } t(F_m) \leq t(B_m) . \end{cases} \quad (2)$$

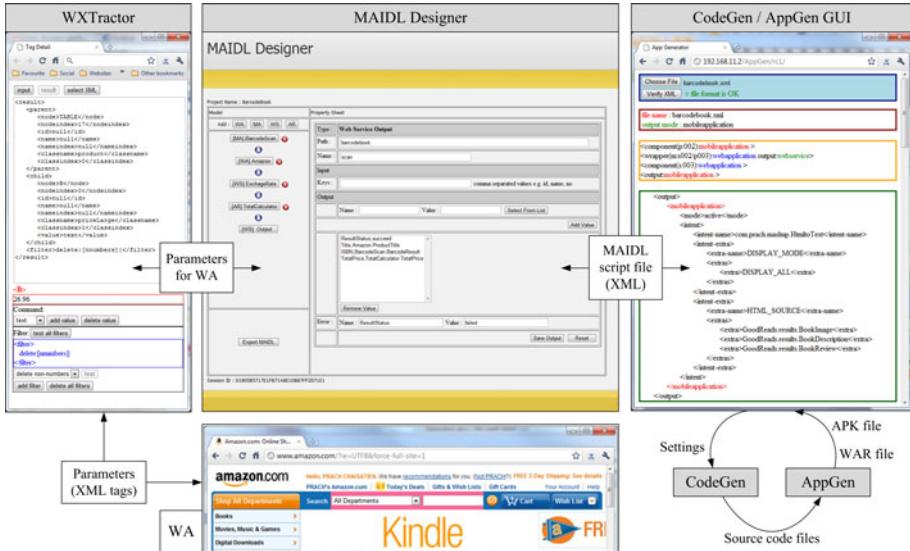


Fig. 4. Screenshots and a workflow in each mashup composition tool

3.4 Mashup Composition Tool

1. *MAIDL Designer* is a Web-based XML document builder, which provides a grammar check and context-sensitive settings. Composers select the output context and then add mashup components. The output of this tool is an XML file (a MAIDL script file), which works with the Code Generator.
2. *Web Extractor* (WXtractor) is a Web browser extension, which generates XML tags used in WA components in MAIDL script files. It will be available for use when composer adds a WA component into his/her mashup. WXtractor provides a visual preview of the annotating part of a Web page (e.g. text content, HTML source code, link URL and image representation if applicable). Composers are also able to specify navigation sequences (e.g. text input, highlight, click) and data filtrations (e.g. delete commas or dollar signs from the text content) using this assistant tool.
3. *Code Generator* (CodeGen) receives a MAIDL script file generated from the MAIDL Designer or manually created by composers. It generates Java source code according to the output context. MA output context's code is generated as files compatible with Android's SDK. The TeWS output context's code is generated as i-jetty [13] files.
4. *Application Generator* (AppGen) is a Web-based compiler, which applies a compilation command to source code generated from CodeGen. It returns a URL link of an mobile application package file (Android package - APK file) or a Web service package file (i-jetty compatible Web archive - WAR file).

4 Evaluation

The evaluation is divided into 3 sections. First, *MAIDL and Mashup Tool* section shows the expressibility test result of MAIDL in composing mashups. Use cases of cooperative applications are given in *Complex Mashup* section. Finally we discuss the mashups' overall *Security Performance* in the third section.

4.1 MAIDL and Mashup Tool

In the evaluation of our tool and description language, we tested the *MAIDL Designer* with 2 subject groups, 5 novice and 6 expert composers. A pre-questionnaire was given to observe the composers' background. After the tool session finished, we also asked composers to fill in a post-questionnaire about the tool's features, their creations, MAIDL's expressibility and their expectations.

The tool session was divided into a tutorial and a freestyle task. The first task was given to let composers use all mashup components (WA, WS, MA and AR) and basic features. The mashup is the one shown in Fig. 1, composers integrated a barcode reader a MA component to read a product barcode, search for the product title and price from *Amazon.com* WA component [14], translate the price currency from dollars to yen using *ExchangeRate* WS component [15], add shipping cost using the AR component ,and finally display the product title and total price via a TeWS. Later, composers were given the freestyle composition task where they could freely plan and create a mashup using our *MAIDL Designer*. We compared the composer's expectations and MAIDL's expressibility by using 5-point Likert scale questions. The data was analyzed using T-test and ANOVA, divided by novice/expert and pre/post task. Finally, the composers asked for comments concerning the mashup runtime and usability.

Composer Groups. Novice composers were able to use the Internet and mobile applications. Our expectation was that they would be able to use our tool to compose mashups with basic functions. We also expected that a similar basic usage pattern amongst novice composers would be found. Expert composers were involved in the use of Web information and are able to program in one or more languages. We expected that they are able to use most of the features provided, suggest corrections and propose more functionality to our system.

Pre-questionnaire Result. The majority of novice composers understand HTML and its simple tags (forms and links). They are familiar with mobile applications but none of them understand the concept of Web services. Novice composers do not know what a mashups are. When asked to give an example of a mobile mashup, they were able to propose ones using GPS and camera functionality as a terminal input. About their expectations for MAIDL (question set C_1), high rating scores were found in: connecting the mashup to a Web service ($\bar{x} = 4.00, \sigma = 1.67$), data filtration ($\bar{x} = 4.00, \sigma = 1.10$), automatic code generation ($\bar{x} = 4.00, \sigma = 1.34$) and creating software package files ($\bar{x} = 4.00, \sigma = 1.73$).

Most of the expert composers understand HTML at the level of tags' attributes. They are familiar with mobile applications and all of them understand

the concepts of Web services and mashups. Mashup examples given by these composers consisted of a wider variety of existent mashup components. Their expectations rated in question set C_1 were low on average since they are highly skilled in programming. High ratings score were found in: connecting to the mashup to mobile applications ($\bar{x} = 3.17, \sigma = 1.60$) and automatic process synchronization ($\bar{x} = 3.67, \sigma = 0.52$). We found low ratings for: the use of mathematical functions ($\bar{x} = 2.00, \sigma = 0.89$) and data filtration ($\bar{x} = 2.00, \sigma = 1.55$).

Tutorial Composition Task. Novice composers finished the task in an average of 34 minutes ($\sigma = 2$). About the expressibility of MAIDL (question set R_1), we found high ratings in these items: planning a workflow using MAIDL ($\bar{x} = 4.00, \sigma = 0.71$) and data filtration ($\bar{x} = 4.00, \sigma = 0.55$). Novice composers are likely to follow the tutorial steps with less details studied.

Expert composers took longer to finish the task with the average of 55 minutes ($\sigma = 27$). In question set R_1 , we found high ratings in: planning a workflow using MAIDL ($\bar{x} = 4.00, \sigma = 0.71$), WA components ($\bar{x} = 3.67, \sigma = 0.52$), WXTractor ($\bar{x} = 3.83, \sigma = 1.60$) and data filtration ($\bar{x} = 4.17, \sigma = 0.75$). From further observations, we found that they took more attention to the reusability of Web applications after WXTractor and the data filter functions were used. Expert composers also tend to spend more time study all features in detail.

Freestyle Composition Task. Novice composers were able to plan the dataflow, choose the right components, and lay out their abstract model using MAIDL. However, advanced features (such as background processing, loop execution) were not used. Therefore, their mashups consisted of only components listed in the manual and time taken in this session was 51 minutes on average ($\sigma = 17$).

The majority of mashups created by expert composers contained 1 to 2 MA components. The complexity of the abstraction models between 2 composer groups was approximately 3 components per a mashup. Differ from the novice composers, expert composers tended to use external Web services not available in the manual as a middle component linking 2 MA components. The average time the group used was 60 minutes ($\sigma = 34$). The reason they took longer might be that they acquired external libraries with custom settings. It takes time to understand how the libraries can be properly configured in MAIDL.

Post-questionnaire Result. A post-questionnaire was taken after the freestyle task finished. Question sets were divided into 3 parts, set C_2 and R_2 which are similar to C_1 and R_1 , and a question set P concerning the composers preference towards MAIDL. In set C_2 , novice composers gave high rating to the reusability of Web applications ($\bar{x} = 4.40, \sigma = 0.55$), simple Web service connections ($\bar{x} = 5.00, \sigma = 0.00$), source code generation, compilation and the creation of software package file ($\bar{x} = 4.80, \sigma = 0.45$). Expert composers gave high ratings to the ability to configure the mashup as a TeWS ($\bar{x} = 4.00, \sigma = 1.55$) and simple reuses of mobile applications as a component ($\bar{x} = 3.83, \sigma = 1.60$). Novice composers' ratings in set R_2 was not significantly high. In contrast, we found high ratings in expert composers' answers emphasizing: planning a workflow using MAIDL, the use of AR components, WA components, and WXTractor ($\bar{x} = 4.50, \sigma = 0.55$).

In set P , we found high novice composers' preference ratings towards: the ability of MAIDL in mashup compositions ($\bar{x} = 4.40, \sigma = 0.55$), the model layout ($\bar{x} = 4.00, \sigma = 0.71$), WXTractor ($\bar{x} = 4.00, \sigma = 1.00$), data filtration ($\bar{x} = 4.60, \sigma = 0.55$) and the publisher's parameter list ($\bar{x} = 4.20, \sigma = 0.84$). Novice composers tended to ignore the use of features involving data type and loop execution ($\bar{x} = 3.20, \sigma = 0.84$). They also did not quite agree that the component library covered all components they needed ($\bar{x} = 3.40, \sigma = 0.55$). They agreed that component's settings should be prepared as templates ($\bar{x} = 4.00, \sigma = 0.71$). The answers from expert composers show high rating in 2 items: the ability of MAIDL in mashup compositions, and the model layout ($\bar{x} = 4.50, \sigma = 0.55$).

Comparison of pre- and post-questionnaire. In novice composer group, a significant difference was not found in the majority of questions in set C and R . However, novice composers might find that the data filtrations are hard to configure as its ratings decreased after all tasks had been finished ($p = 0.03$). Expert composers might be able to reduce their efforts to write the components' code manually as a significant increase in rating are found in these items: the use of WA components ($p = 0.02$), and the use of AR components ($p = 0.01$).

Runtime and Usability. The runtime performance of the applications created by novice composers was normal when running MA components and WS components without complex queries. One application, which connected 2 Web applications, was slow and sometimes became unresponsive due to high network traffics. Most of the applications created in the expert composers freestyle task contained one WS component as a middle component and had a faster runtime. However, when an expected query result of Twitter API [16] was not found, the mashup process halted and the application needed to be restarted. In MAIDL, there is still no conditional statement for deciding processes or skip errors. Expert composers suggested that status and dialog configurations should be included.

Interpretation and Conclusion. Using MAIDL, novice composers agree that they are facilitated with simple configurations to reuse Web information. After the tutorial task, they were able to abstract the model of their mashup. WXTractor is able to assist their data extraction from Web applications. However, advanced features such as data type and loop executions were not preferred. The code generator and compilation tools could assist their lack of coding knowledge. They tended not to use external libraries, therefore, component libraries and templates should be added. The majority of novice composers also believe that MAIDL enables easy mashup compositions and opens opportunities for non-programmers and new comers to this area. More visualizations should be used and they also suggested that the look and feel of the mashup should be customizable.

Expert composers' expectation towards MAIDL's expressibility was low in the first place and increased after they finished the tasks. Composers in this group tended to use more time in the tasks because they were trying to adapt their skills (e.g. applying Web applications and Web services). The expressibility and customizability of MAIDL met their needs when applying it to external libraries. However, they suggested that some automatic functions (e.g. data adaptations,

component templates) should be implemented or manually addable. In addition, expert composers found that the use of WXTractor to annotate tags and apply filters to them was easy. They suggested that it would be helpful for non-programmers if the same method was applied to XML or JSON messages. Just as novice composers did, expert composers suggested that the *MAIDL Designer* have more visual context-aware GUIs and more component templates. A high customizability in MAIDL was good for them to apply external libraries but might not be appropriate for non-programmers. If possible, data adaptations should be done automatically when the data are sent to the component.

Comparison between Novice and Expert Composers. T-test and ANOVA was applied to C_1 , C_2 and R_1 , R_2 data pairs of novice and expert composers which can be interpreted as follows:

1. Expectations of the expert and novice groups were met after the evaluation.
2. The novice group has higher expectations towards MAIDL than the experts.
3. The expert group rated expressibility of MAIDL in assisting mashup compositions higher than the novice group.
4. After the evaluation was finished, the expert group agreed that MAIDL gave higher rating towards expressibility in assisting mashup compositions.

The result can be interpret that MAIDL might not perform well when mashups are composed by novice composers because of its complexity. Expert composers are able to use MAIDL without confusion and may apply it to external libraries. However, both groups' expectations are met. Composers in both groups rated that the approach delivered 75% subjective rating for creating mashups.

4.2 Complex Mashup

To study cooperative mashup applications, we created 2 complex mashups using our approach. The mashups require interaction between 2 or more mobile devices. In this way, the mashup created in a TeWS output context is deployed on an Android device. On the other hand, iOS devices [17] are manually programmed to consume the deployed TeWS. We evaluate their runtime, connection performance and interaction usability on each device in the actual running settings.

Meeting Point: Cooperative Geolocation Mashup. In this mashup, geolocation of 2 devices are used as a data to find a list of restaurants located near the middle point between each device's GPS coordinates (via the *GourNavi* Web service [18]). Fig. 5 shows 2 mashup models and mashup applications, *Meeting Point Registration* and *Meeting Point Confirmation*, which communicate between devices via a TeWS in separated contexts.

Internal Runtime and Connection Performance. Application on the iOS side was presumably lightweight. Since this is a cooperative mashup for 2 devices with handshaking-like protocol, multiple connections are not considered as a performance factor. The overall performance of this mashup depends on the performance of *GourNavi* Web service.

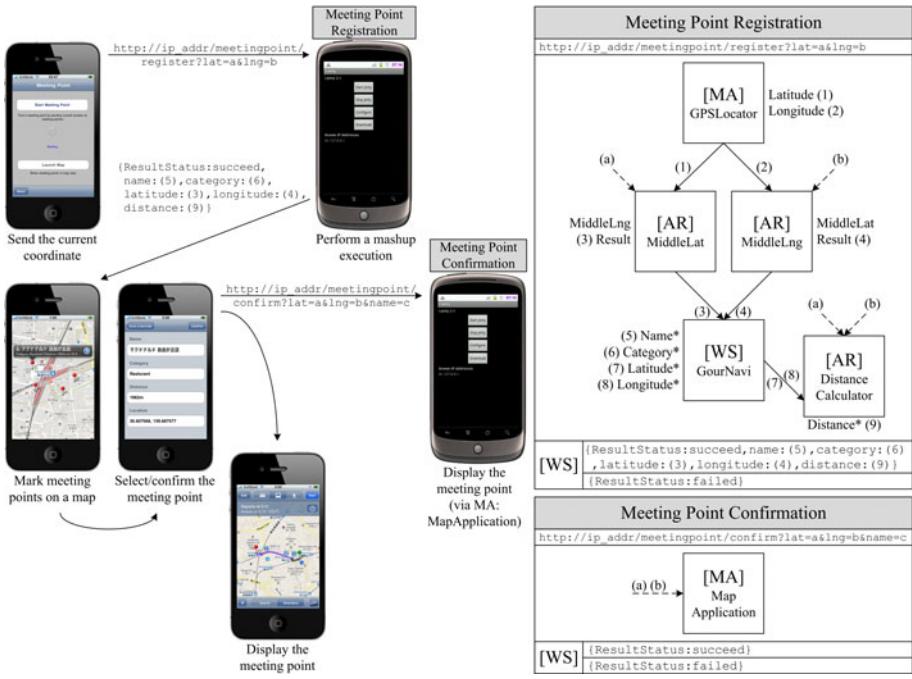


Fig. 5. Mashup models and screenshots of *Meeting Point*

Usability and Interactions. If we assumed that 2 devices are connected using global IP addresses and are placed outdoors, the interactions between 2 devices might be interrupted by signal loss. Both sides must have timeout configuration and reconnection arrangement in the case of failure execution.

4.3 Book Shopping: Camera and Data Server Cooperative Mashup

This complex mashup application is designed for a shopping scenario in a book store for 2 or more users. One user holds an Android phone functioned as a server, the other users are holding iOS devices and are moving around the store searching for books. Mashup applications in a TeWS output context consist of *Book Shopping Add* and *Book Summary*. iOS device clients were installed with a manually written program to read a book's barcode and send the translated data to the *Book Shopping Add* TeWS deployed on the Android phone. The TeWS on the phone will search for the product title and price from *Amazon.com* WA component, translate the currency of the price from dollars to yen using *ExchangeRate* WS component, and finally add the title and price in Japanese yen into the phone database. Users with an iOS device may request for the added book title, book price and price summary using the *Book Summary* TeWS.

Internal Runtime and Connection Performance. In this setting, 2 mashups are installed on the server and handle multiple connections. In *Book Shopping Add*,

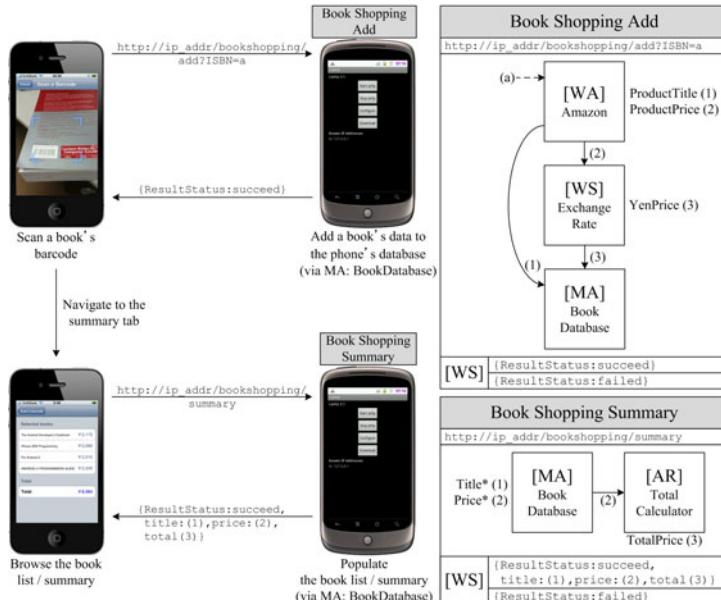


Fig. 6. Mashup models and screenshots of *Book Shopping*

the overall performance depends on the *Amazon.com* WA component and the *BookDatabase* MA component. We implemented 2 versions of *BookDatabase*, foreground and background ones. In *Book Shopping Summary*, when the MA component is accessed as a foreground process, server connections are queued by MAIDL process synchronization feature. The mashup runs more smoothly when the composited MA components are accessed as a background process. However, the WA component in *Book Shopping Add* runs in foreground process, yields lower speed and the server queues all incoming connections. Therefore, this type of mashup should not include foreground WA and MA components.

Usability and Interactions. We have tested the actual mashup using wireless network connection and found that the range between the server and client devices are important. This mashup might also be applied to other shopping scenarios which are best suited for Ad-hoc connections.

4.4 Security Performance

Since the application of WA components in MAIDL to invoke cross-domain connections and navigations of secured Web sites are possible. We believe that users should be warned before the mashup is installed on the device. For MA components, faulty MA components, which enter infinite loops, are automatically terminated by Android's system. However, the endless loop of sending data in a circle can be implemented. The use of MA component in mashup applications also altered some security issues. Android's OS allows softwares to be removed

through the Intent messaging protocol. To solve these problems involving insecure WA and MA components, a MAIDL script might also work as a manifest file to look for information of the integrated component for security reasons.

The other important issue is when a mashup called a MA component which accessed personal information. In general, a manifest file of a mobile application is used to indicate what function to be used and what information to be accessed. Therefore, security measurement of each component in a mashup should be conducted and informed to the users before the mashup is installed and used.

5 Discussion

The results of the evaluation indicate that our method provides a good solution to mobile mashup compositions. The composers emphasized that the use of visual previews of data extractions and filtrations are preferred. However, the system do not allow mashups be simulated as a whole. Some MA components required an actual runtime environment. Moreover, we believe that mobile application providers or online application stores should contain the application's description concerning the messaging protocol for mashups.

In general, a mobile mashup is created for one task. Compared to mashups of Web applications, we found that mobile mashups have less complexity. They tend to be created as an integration between the phone's sensors and Web information. We found fewer mobile mashups that perform comparisons of multiple lookups to a variety of Web sites. More factors have to be considered in creating mashup in TeWS output context. To deliver smooth interactions between devices, the behavior of running process, network latency and usage scenario has to be observed. Since MAIDL script files contain information about each component and its runtime behavior, an alternative application of MAIDL for performance measurement can be considered.

MAIDL script files also contain a concrete description of the output messages sent via a TeWS. Applications on the client side might be generated or adapt themselves according to the description. A good example for the combination of a TeWS and a desktop Web application is to exchange multiple data from a mobile phone to automatically fill in personal information in an HTML form. The Web application first observes the applicable TeWS on the device and connects to it.

For the power consumption of the mashup, energy footprint of each component have to be observed, including the amount of data sent via a TeWS.

6 Conclusion and Future Work

In this research, we proposed a fast-paced mashup development using MAIDL. The mashup created by our approach can be designated for a single device, as a normal mobile application, or for multiple devices, as a TeWS. We proposed the method of *Mashup Output Context Transformation* and *Mashup Process Scheduling Algorithm*. The composition enables integration of annotated parts of Web pages, connections to Web services and the use of existent mobile applications.

In the evaluation with novice and expert composer groups, we found that the approach's simplicity in reuse of Web applications, Web services and mobile applications as mashup components serve the composers well. For the approach's expressibility, novice composers tend to use normal features and pre-defined templates, while expert composers require customizability when applying external libraries. Both composers groups gave higher preference rating after they used our tool. This implies that our method is optimal for low skilled composers.

In the complex mashup section, we demonstrated how a mashup works in a TeWS output context to deliver functionality exchange and cooperative application between devices. We were able to characterize the components that are appropriate to be integrated in a complex mashup in TeWS context. Our future work is to enable mobile mashup in the context of a Web application on a mobile device. To support higher interactivity to run on desktop computers, the process control and composition method might be different from the two contexts we have observed. The combination of multiple TeWS might be considered.

References

1. Appcelerator Titanium Mobile, <http://www.appcelerator.com/products/titanium-mobile-application-development/>
2. PhoneGap, <http://www.phonegap.com/>
3. Pietchmann, S., Tietz, V., Reimann, J., Liebing, C., Pohle, M.: Meißner, K.: A Metamodel for Context-Aware Component-Based Mashup Applications. In: Proceeding of the 12th International Conference on Information Integration and Web-Based Applications & Services. ACM, New York (2010)
4. Chaisatien, P., Tokuda, T.: A Description-based Approach to Mashup of Web Applications, Web Services and Mobile Phone Applications. In: Information Modelling and Knowledge Bases XXII, Frontiers in Artificial Intelligence and Applications, vol. 225, pp. 174–193. IOS Press, Amsterdam (2011)
5. Paternò, F., Santoro, C., Spano, L.D.: Maria: A universal, declarative, multiple abstraction level language for service-oriented applications in ubiquitous environments. In: Computer-Human Interaction, vol. 16 (2009)
6. Ajijaz, F., Ali, S.Z., Chaudhary, M.A., Walke, B.: The Resource-Oriented Mobile Web Server for Long-Lived Services. In: 6th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (2010)
7. Google App Inventor for Android, <http://appinventor.googlelabs.com/>
8. Kaltofen, S., Milrad, M., Kurti, A.: A Cross-Platform Software System to Create and Deploy Mobile Mashups. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 518–521. Springer, Heidelberg (2010)
9. Maleshkova, M., Pedrinaci, C., Domingue, J.: Semantic Annotation of Web APIs with SWEET. In: Proceedings of the 6th Workshop on Scripting and Development for the Semantic Web (2010)
10. Guo, J., Chaisatien, P., Han, H., Noro, T., Tokuda, T.: Partial Information Extraction Approach to Lightweight Integration on the Web. In: Daniel, F., Facca, F.M. (eds.) ICWE 2010. LNCS, vol. 6385, pp. 372–383. Springer, Heidelberg (2010)
11. Android Developers, <http://developer.android.com/index.html>

12. Android Intents, <http://developer.android.com/guide/topics/intents/>
13. i-jetty, <http://code.google.com/p/i-jetty/>
14. Amazon.com, <http://www.amazon.com/>
15. Exchange Rate API, <http://www.exchangerate-api.com/>
16. Twitter Search API, <http://dev.twitter.com/doc/get/search>
17. iOS Technology Overview, <http://developer.apple.com/technologies/ios/>
18. Gourmet Navigator API, <http://api.gnavi.co.jp/api/manual.htm>

A Framework for Concern-Sensitive, Client-Side Adaptation

Sergio Firmenich^{1,2}, Marco Winckler³, Gustavo Rossi^{1,2}, and Silvia Gordillo^{1,4}

¹ LIFIA, Facultad de Informática,

² Universidad Nacional de La Plata and Conicet Argentina

{sergio.firmenich,gordillo,gustavo}@lifia.info.unlp.edu.ar

³ IRIT, Université Paul Sabatier, France

winckler@irit.fr

⁴ CiCPBA

Abstract. Currently the Web is a platform for performing complex tasks which involve dealing with different Web applications. However, users still have to face these tasks in a handcrafted way. While building “opportunistic” service-based software, such as mashups, can be a solution for combining data and information from different providers, many times this approach might have limitations. In this paper we present a novel approach which combines concern-sensitive application adaptation with user-collected data to improve the user experience while performing a task. We have developed some simple though powerful tools for applying this approach to some typical tasks such as trip planning. We illustrate the paper with simple though realistic examples and compare our work with others in the same field.

1 Introduction

As wisely pointed out in [6], one of the most interesting facets of Web evolution is the kind of end-users interaction with Web contents. At first, users could only browse through contents provided by Web sites. Later, users could actively contribute with content by using tools (e.g. CMS, wikis) embedded into these sites. More recently different technologies provide users with tools allowing them to change the way Web content was presented. For example, using visual Mashups [5, 14], users can compose content hosted by diverse Web sites and they can run Greasemonkey scripts [9] to change third part Web applications by adding content and/or controls (e.g. highlight search results in Amazon.com which refer to Kindle).

These tools built under the concept of *Web augmentation* [2] extend what user can do with Web contents, but they provided limited support to tasks that require navigation on multiple Web sites. For example, a user who is using the Web for planning a holiday trip to Paris might ultimately visit several sites such as *expedia.com* for flights, *booking.com* for hotels, *wikipedia.org* for general information about the city and *parisinfo.fr* for points of interest, current events or expositions in Paris. From the users’ point of view, the navigation of all these sites is part of the same task. The existing augmentation techniques are of little help in this case. For example, GreaseMonkey scripts can adapt the content on a specific Web site but it will require much

effort to make it generic enough to integrate information provided by different applications. Mashups, meanwhile, can be used to integrate content from several Web sites; however, a Mashup for *expedia.com* will not necessarily integrate information from other users' preferred Web sites (e.g. *airfrance.fr*, *venere.com*...). If these sites provide public APIs, Mashups can be extended, but it does not prevent users to learn how to do it beforehand. Quite often, users' tasks are associated with opportunistic navigation on different Web sites, which is difficult to predict [12]. In this context, effective Web augmentation should overcome two main barriers: i) to take into account different applications which are visited by users (either through explicit navigation or just opening a new browser's window with the corresponding URL); and ii) to adapt the unknown target Web sites, considering that the user might need different kind of adaptations at different sites.

This paper proposes a framework for creating flexible, light-weight and effective adaptations to support users' tasks during the navigation of diverse Web applications. Our goal is to support users' tasks by keeping his actual concern (and related data) persistent through applications. For example, allowing that dates used on *expedia.com* for booking a flight could be reused as input for *booking.com* while booking hotels in the same period. Another example of adaptation that illustrate our approach is the inclusion of new links allowing users to easily navigate from *parisinfo.fr* to related articles at *wikipedia.com* whenever he needs further explanation about a topic.

In a previous work [8], we showed how to profit from the knowledge of the current user's concern to improve navigation in Web applications, by enriching the target page with information or links which are useful in that specific concern. In this paper, we push further this approach to allow adaptations that go beyond a single application's boundary. Moreover, we present a framework and a set of tools which allow simplifying the process of concern-sensitive Web augmentation, reducing the programming burden, and therefore allowing end-users to configure their own adaptations even when they are complex as in the example above.

This paper is organized as follows: in section 2 we provide an overview of related work. The framework is fully described in section 3. Section 4 presents tools built upon the framework. Section 5 presents how we have validated our approach with end-users. Finally, section 6 present conclusions and future work.

2 Related Work

The field of Web applications adaptation is broad; therefore, for the sake of conciseness we will concentrate on those research works which are close to our intent. The interested reader can find more material on the general subject in [4]. As stated in the introduction we can identify two coarse-grained approaches for end-user development in Web applications: i) mashing up contents or services in a new application and ii) adapting the augmented application, generally by running adaptation scripts in the client side.

Mashups are an interesting alternative for final users to combine existing resources and services in a new specialized application. Visual and intuitive tools such as [5, 14] simplify the development of these applications. Since most Web applications do not provide Web services to access their functionality or information, [10] proposes a

novel approach to integrate contents of third party applications by describing and extracting these contents at the client side and to use these contents later by generating virtual Web services that allow accessing them.

The second alternative to build support for users tasks is Web augmentation [2], where the target application is modified (adapted) instead of “integrated” in a new one. This approach is very popular since it is an excellent vehicle for crowdsourcing. Many popular Web applications such as Gmail have incorporated some of these user-programmed adaptations into their applications, like the mail delete button (See <http://userscripts.org/scripts/show/1345>). The most popular tool to support Web augmentation is GreaseMonkey [9], whose scripts are written in JavaScript. The problem with these scripts is their dependence on the DOM; if the DOM changes the script can stop working. In [6] the authors propose a way to make GreaseMonkey scripts more robust, by using a conceptual layer (provided by the Web application developer) over the DOM. In [7] the authors extend the idea to allow scripts developers to write their own conceptual abstractions to cut the dependency with unknown developers; in this way, when the DOM changes, the maintenance is easier because only the matching between the concepts and the DOM need to be redefined.

While we share the philosophy behind these works, we believe that it is necessary to go a step further in the kind of supported adaptations. In [8] we showed how to use the actual user concern (expressed in his navigational history) as an additional parameter to adapt the target application. By using the scripting interface we managed to make the process more modular, and by defining adaptations for application families (e.g. social networks) we improved the reuse of adaptation scripts. In the following sections we show how to broaden the approach allowing end users to select which information can be used to perform the adaptation, therefore improving the support for his task and providing support for building more complex adaptations.

3 A Framework for Concern-Sensitive Augmentation

For the sake of comprehension we first introduce some basic concepts and background work; next we make an overview of the approach and of our tool support.

3.1 Background for the Framework

Our framework is based on the concept of concern-sensitive navigation (CSN). We say that a Web application (or specifically a Web page) is concern-sensitive (CS) when its contents, operations and outgoing navigation links can change (or adapt) to follow the actual situation (concern) in which it is accessed [8]. Concern-sensitive navigation is different from context-aware navigation, where other contextual parameters (location, time, preferences) are considered. Figure 1 illustrates the differences between flat and concern-sensitive navigation. Note that there are two kinds of navigations: Flat navigations (represented with solid arrows) where the target Web pages show always the same information, without taking into account the source of navigation; in concern-sensitive navigations (represented with dashed arrows) meanwhile, the target pages adapt or enrich their contents by taking into account what was the user concern in the previous page.

In [8] we have argued that concern-sensitive navigation simplifies the user's tasks by providing him sensitive information or options according to his current needs. We have also introduced an approach to build smart client-side adaptations, implemented as browsers' plugins, which allow making specific Web applications aware of the concern in which they were accessed, changing contents and links in consequence.

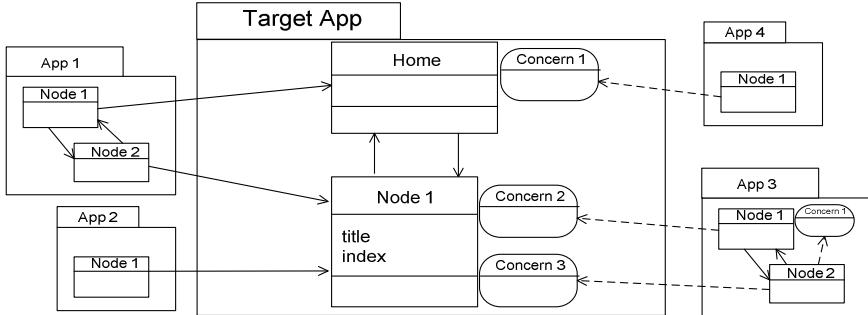


Fig. 1. Flat Navigation vs. Concern-Sensitive Navigation

Figure 2 shows an example of concern-sensitive navigation across two applications: Google Maps (as the source of navigation) and Wikipedia (as the target). The left-side displays Wikipedia links in the map of Paris; once selected, these links trigger the page at the right-side of Figure 2, augmented with the corresponding map and a set of links to those Wikipedia articles in the surroundings of the current one.



Fig. 2. Inter-application CSN between Google Maps and Wikipedia

In general, the task of CS adaptation of a page P requires that we: (a) know the actual user's navigation concern (i.e. pages previously navigated, e.g. Google maps), (b) record the set of relevant information from previously visited pages that are needed for adaptation (e.g. the current map), and (c) have the capacity for enriching P with contents or links related with (a) and (b) by intervening in P 's DOM.

3.2 The Approach in a Nutshell

The CSN approach works well for application families (e.g. plugins that work for similar applications which share some features). However, it "only" provides

end-users with a fixed set of adaptations. We have developed a software framework which extends the concept of CSN by providing different kinds of users (end-users, developers, etc) a set of tools to augment Web applications by considering the actual user concern. Developers can use the framework to implement new adaptation functions, named *augmenters*. Augmenters are built as generic adaptations featuring behaviours such as *automatic filling in forms*, *highlighting text*, etc. End-users can benefit of these *augmenters* during navigating by “collecting” concern information to be used when adapting the user interface (See section 3.3.1). By combining augmenters, the framework also supports *scenario engineering* for developing customized adaptations for specific domains such as trip planning (See section 3.3.2). For example a scenario can be based in the use of the form filling augmenter when the user is navigating among several Web sites for booking flights and hotels. The same form filling augmenter can be used to fill forms related to a product search in different e-commerce Web sites, for example by taking the department (e.g. *electronics*) and the keyword (e.g. *iphone4*) used in *amazon.com* to complete the form automatically in *fnac.fr*.

The framework is described at Figure 3 using the pyramid approach [11]. The top levels are more abstract while lower ones are more detailed. At the top layer, final users can collect relevant information for their current task or concern by using the *DataCollector* tool. Then, when they navigate to other sites they are able to execute augmenters using this information; in this way they can satisfy volatile requirements of adaptation (not foreseen by developers). At the middle layer, end users with programming skills can extend the framework by developing augmenters and scenarios as classes inheriting of *AbstractAdapter* and *AbstractScenario*, two outstanding framework hot-spots. The bottom layer shows a more detailed view of the framework design; a third hot-spot, *AbstractComponent* abstracts concrete components used in scenarios; for example we developed a component which offers geo-location information; another tool could empower the scenarios by giving them auto fill forms capabilities (e.g. a component that implements carbon [1]).

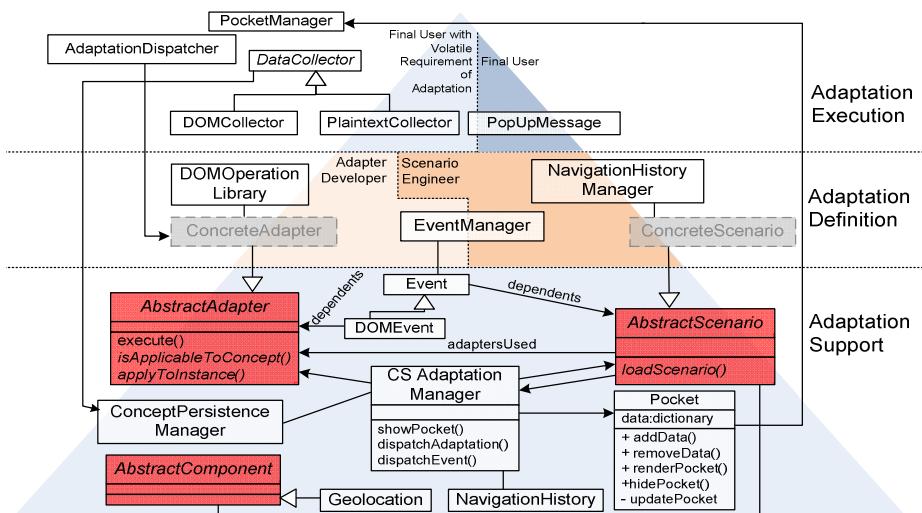


Fig. 3. Framework structure

Framework components act like libraries to be used for developing adaptations. We briefly outline the main framework components:

- **Adaptation Support Layer**

- ClientSideAdaptationManager: is the Framework's core, whose functions are to coordinate others elements and to serve as communicator with the browser.
- NavigationHistory: is the navigation history object provided by the browser. We have developed a wrapper on top of it to ease scenarios development.
- ConceptPersistenceManager: is responsible for saving and restoring user data into the local files system.
- AbstractAdapter and AbstractScenario: are abstract classes from which concrete augmenters and scenarios, correspondingly, developed by users must inherit.
- AbstractComponent: is an abstract class used for extending the framework by developing components to support new capabilities (e.g. geolocation).

- **Adaptation Definition Layer**

- DOMOperationLibrary: a library that operates with DOM elements; it raises the level of typical JavaScript sentences easing the development of augmenters.
- EventManager: is the responsible of adding and removing listeners (Adaptation Definition Layer) of events from the lower layer.
- NavigationHistoryManager: is a wrapper with which scenarios can make queries about navigation history.
- ConcreteAdapter and ConcreteScenarios: are scripts developed by users with programming skills. These classes are shown in Figure 3 in order to highlight their place in the hierarchy. Some concrete augmenters as HighlightAdapter, WikiLinkConverter, CopyIntoInputAdapter are included in our framework.

- **Adaptation Execution Layer**

- DataCollector: is the tool to allow users collecting information while navigating. So far, two concrete DataCollectors have been implemented: one for selecting plaintext information, and another to handle DOM elements.
- PocketManager: is our tool to allow users to move information among sites.
- AdaptationDispatcher: is the responsible of executing an adaptation under user demand. It is useful to accomplish volatile requirements of adaptation.

3.3 Extending the Framework

The framework can be extended in two ways: by creating new augmenters (generic basic adaptations), and by building scenarios (for supporting specific user's tasks). Although we do not restrict the kind of adaptations, we fully support the development of adaptations which take into account the actual user concern. Since many times it is not enough to be aware of the user's navigation history to fully know his concern, further information about his current activity is often needed. The example given at Figure 2, shows how some information is moved from *GoogleMaps* to *Wikipedia*. Our framework offers two kinds of tools to move information among Web sites. The first one is the *DataCollector* with which users can select elements from the current Web page. The elements selected are added into the second tool named *Pocket* which can

store either simple plain text or data with some semantic meaning as a concept name. Once the information is stored into the *Pocket*, it will remain available for any Web pages visited later on. Section 4.1 details how users collect information during navigation.

3.3.1 Creating Augmenters with the Framework

The simplest way to extend our Framework is to develop a new augmenter. An augmenter is an adaptation component developed by users with programming skills. Augmenters have two main contributions in our adaptation approach: they provide tools for satisfying end-users' volatile requirements for adaptations and they support the development of sophisticated scenarios built by combining simpler augmenters.

An augmenter can be standalone or be executed with data collected as argument; in this case this data is assigned by the actor who triggers the augmenter execution (either a scenario or the user). For example, an augmenter aimed to highlight elements in the page, must be able to do it for an element (for example the *City* instance “Paris”) or for a collection of elements (for example, all *City* instances). Therefore augmenters should be flexible with regard to the user's needs. Figure 4 shows an augmenter (*WikiLinkConversion*) applied to *parisinfo.com* with the user coming from *wikipedia.com* with his *PointOfInterest* instances (these are strings collected from the Web pages visited and conceptualized or typed as *PointOfInterest* in the *Pocket* (the floating box showed at right in the Figure). As Figure 4 shows, the augmenter *WikiLinkConversion* is applied to any *PointOfInterest* occurrence in the page. Note that when the user right clicks over *PointOfInterest*, a menu with the available augmenters is opened and then he chooses “Convert to Wiki Link”, so *WikiLinkConversion* is executed with all instances of *PointOfInterest* as parameters.



Fig. 4. Plain text converted into links to add personal navigation

Since augmenters can be applied to different Web pages they must be developed without a dependence of a particular DOM, as described in [6]. Moreover, when using the framework, developers must:

- Construct an augmenter as a JavaScript object inheriting from *AbstractAdapter*, the hot-spot shown in Figure 3.
- Implement the methods defined as abstract in *AbstractAdapter*. This is necessary because the *execute()* method of *AbstractAdapter* (a template method) sends messages to concrete augmenters. Since the method *execute()* is the starting point of an augmenter, if a message can not be dispatched, the execution will fail. The method *execute()* receives data as parameter which is used to perform the adaptation.

Manipulating the DOM to adapt the page is a responsibility of augmenters. Since DOM manipulation can be hard for users, the framework provides them with the *DOMOperationLibrary*, a component inspired in the most popular JavaScript libraries like Prototype (see <http://www.prototypejs.org/>) and jQuery (see <http://jquery.com/>) to make DOM manipulation simpler. In this way, target DOM elements (those that are abstracted by elements from the *Pocket*) are easily manipulated by operations like style changes, hiding, removing, or adding content.

Augmenters are executed when a user explicit triggers them or when a scenario is instantiated (see section 3.3.2). In Figure 5.a, we show a sequence diagram to demonstrate how the framework chains the execution of augmenters. The object *User* represents the real user. First, the user chooses an element from the *Pocket* and when he right clicks over it; a menu is opened with all augmenters available. When he selects one of them, the *Pocket* sends the *dispatch()* message to the *AdaptationDispatcher* that finally executes the augmenter with the *execute()* message. Note that when an augmenter receives the *execute* message, it sends to itself both the *isApplicableToConcept* and *applyToInstance* messages. All augmenters developed by users must have these methods defined as in the augmenters showed in Figure 5.b

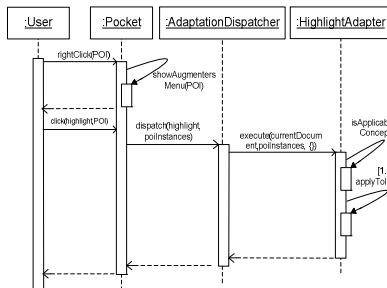


Fig. 5. a. sequence diagram describing user triggering an augmenter.

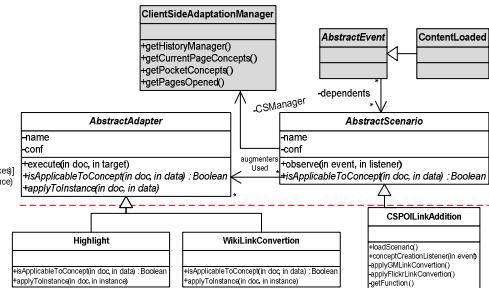


Fig. 5. b. class diagram presenting the framework extensions with augmenters and scenarios.

3.3.2 Creating Scenarios with the Framework

Augmenters are useful to perform simple tasks on a site; however, for complex tasks users perform sets of activities, many times following pre-defined patterns. For example, booking flights, and then booking a hotel is a common scenario. In different moments (and moreover for different users) the Web pages used to do these tasks may change. However, the information used during the task is similar and the kind of adaptation needed too. For example depart date, arrival date, and a destination are all the pieces of information needed to perform (in a simplified view) this task in any Web site of this kind.

A scenario is an event-driven script; it registers listeners for those events in which it is interested in. These events usually refer to the user activity as when he opens sites or collects new data. When an event occurs the scenario is loaded and it first checks that the information it needs is available; if so, the scenario is instantiated. Scenarios execute adaptations when some conditions (e.g. about the navigational history or

collected data) are satisfied; to perform adaptations, they trigger augmenters that change the DOM. A scenario could execute the same augmenter, but with different arguments as Figure 6 shows. In Figure 6.a a Wikipedia article is adapted in the context of a scenario. The scenario uses the LinkAdditionAdapter (an augmenter similar to the one described in the previous section) to add a link close to each occurrence of the *PointOfInterest* concept. In Figure 6.a, the target elements are all instances of the *PointOfInterest* concept, and the adaptation is executed automatically when Flickr.com appears in the navigational history. Figure 6.b shows a similar case, but now since GoogleMaps is the previously visited Web page, a link to GoogleMaps is added.

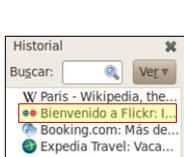


Fig. 6. a. Navigation with Flickr concern.



Fig. 6. b. Navigation with GoogleMaps concern.

A scenario is realized in a quite similar way than augmenters (in the sense of being a JavaScript file) but with some distinct features to register its interest in different events. The scenario engineer has to respect these constraints:

- Construct the scenario as a JavaScript object inheriting from *AbstractScenario*, the hot-spot shown in Figure 3.
- Implement the methods defined as abstract in *AbstractScenario*. There are methods that will be executed during initialization when the browser is opened. Note, for example, that scenarios can be interested in different events; therefore they must register listeners which will be executed in order to instantiate the scenario when the events happen. The same kind of inversion of control occurs when the framework sends the *loadScenario()* message in order to wakeup the scenario.
- Specify which augmenters are necessary to carry out the scenario.
- Specify the set of concepts needed to instantiate the scenario and define them in the *DataCollector* tool; thus, when users collect data, the available concepts or types are those in which the scenarios are interested in (e.g. destination, dates, etc.).

A scenario needs to manage more information than an augmenter. In this sense a Scenario Engineer can use some tools provided by our framework that give him:

- The capability to add listeners to different events which will take place in the user navigation context. For example a scenario could express interest in a Web page load (*contentLoadedEvent*), or even in the instantiation of some particular concept (*cityInstantiatedEvent*); this event occurs when the user has added a particular value typed as City into the *Pocket*.
- Knowledge about the navigation history.
- Knowledge about concepts and concepts instances stored into the *Pocket*.

Scenarios are not magically executed. A scenario is latent, waiting for the signal needed to be executed. For example, the *destinationInstantiated* event could trigger the scenario if it had registered a listener to be executed when instances of the Destination concept are created (see an example in section 4.3). To illustrate this, in Figure 7 we show how a scenario is executed when the user opens a Wikipedia article. At the left of this Figure, we show a sequence diagram for the scenario CSPOILLinkAddition, a concrete scenario for the example of Figure 6. First, the scenario adds itself as the listener of the *contentLoadedEvent*. Then, once the content is loaded, the EventManager object loads all scenarios that are waiting for this event (in the example there is only one scenario). In the example of Figure 7, CSPOILLinkAddition consults the NavigationHistoryManager to know if the previous node of the history is GoogleMaps and, as it is true, CSPOILLinkAddition sends the *applyGMLinkConvert* message. The method *applyGMLinkConvert* gets all instances of the concept *PointOfInterest* by sending the message *getAllInstances* to the *Pocket* object. After that, it sends the message *execute* to the augmenter (*LinkAdditionAdapter*) with the current document (it is the DOM target), all the *PointOfInterest* instances and a dictionary with parameters that the augmenter needs.

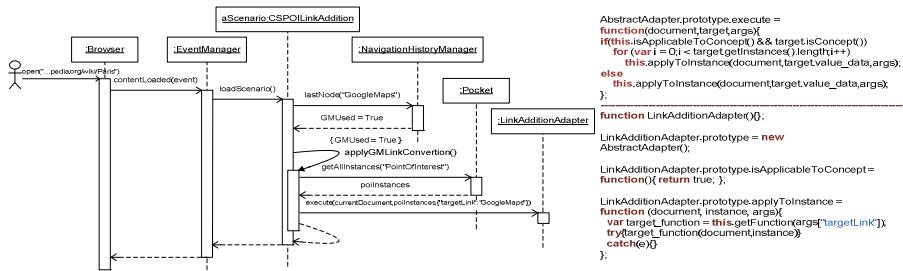


Fig. 7. Sequence diagram for a scenario execution and code of the augmenter applied

The right side of Figure 7 shows an excerpt of the augmenter code used in this scenario. The method *execute()* of the *AbstractAdapter* is first shown. This is a template method that sends both *isApplicableToConcept* and *applyToInstance* messages, which are defined in *LinkAdditionAdapter*. This augmenter has others method like *getFunction* that are not shown by the sake of conciseness.

4 Tool Support

The framework was implemented as a Firefox extension that provides all components shown in the pyramid of Figure 3, plus other components such as some defaults augmenters. Hereafter, we illustrate the use of augmenters and scenarios by end-users.

4.1 Data Collector

In our approach, end-users execute the adaptations. Although this can be made both explicitly (when users execute some augmenter) or implicitly (when a scenario is

instantiated), some information is always needed since we aim to improve the user experience by adapting the Web pages he navigates according to the needs of his current concern. In this way users are empowered with tools to (when necessary) collect meaningful information while they visit Web sites. This information can be collected “automatically” when the user is instantiating a previously developed scenario, and the underlying tool is aware of the semantics of the pages’ data, or might be collected “by hand” using tools provided by the framework (concrete *DataCollectors*). A *DataCollector* allows users to define untyped data (in order to quickly add information into the *Pocket* for volatile adaptations), and typed data (usually to add information for scenarios). The information collected is later available into the *Pocket* and it can be used to perform adaptations. In Figure 8.a the user stores different information elements, collected with the *PlaintextCollector* component, into the *Pocket*. *PlaintextCollector* has two options, “Put into the Pocket” and “Put into the Pocket as volatile data” as it is shown. In this figure he collects several points of interest that he would like to visit (from the Wikipedia article) and keeps them in the pocket. Since he wants to type them as “*PointOfInterest*” he uses the “Put into Pocket” option which opens the dialog.

4.2 Description of Default Augmenters in the Framework

Currently, some augmenters are provided by default with the framework. Some remarkable ones follow:

- *Highlight*: it highlights the occurrences of the data received by parameter.
- *CopyIntoInput*: it pastes the value received as parameter into an input form field. Once the augmenter is executed, it adds a listener to the click event which is removed after the first time in which the target is an input.
- *WikiLinkConversion*: it creates links to *wikipedia.com* pages using as input any occurrences values received as a parameter. For example if the parameter is “Paris” then the link would be to the Wikipedia article about Paris.

The augmenters *Highlight* and *WikiLinkConversion* can perform adaptations for a single value (e.g. “Paris”) or for a collection of values, instances of a concept (e.g. City). The augmenter *CopyIntoInput* can only be executed with a single value.

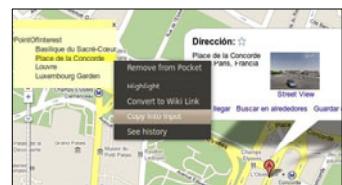


Fig. 8. a. Information extraction from Wikipedia

Fig. 8. b. Resulting adaptation

As an example we show how the *CopyIntoInput* augmenter is used in Figure 8.b. Here the data collected before as instances of “*PointOfInterest*” is available into the

Pocket (the floating box at the left in Figure 8.b) when the user opens Google Maps. In order to use one of these instances the user right clicks over the target point of interest to open the contextual menu with the augmenters available for the current site. Once he has chosen the *CopyIntoInput* augmenter, the *AdaptationDispatcher* triggers it. Since the augmenters provided as defaults are generic, they will always appear in the contextual menu. However which augmenters are available depend on the current site because augmenters can be generic enough to be applied to any page (e.g. highlight) or specific for a single site (e.g. search the target value as a location in GoogleMaps).

4.3 Scenario Instantiation by End-Users

A scenario waits for an event to be loaded; when the event occurs and all conditions are satisfied, it is instantiated. Figure 9 shows the initial steps that a user would possibly perform to satisfy the needs previously described at section 1. In Figure 9.a, while he books (or just explores) the flights to Paris, he collects some data which will be useful in the following steps. When users collect data they can give them a conceptual meaning, by assigning a type to the selected value. In the example, the types used are *departDate*, *arriveDate* and *destination*. When some data is collected the corresponding event is triggered (e.g. *destinationInstantiated*). In Figure 9.a we show how the scenario shows a popup message to offer users to use the information collected for booking hotels; this message is showed after the dates and destination were collected. Figure 9.b shows how the form field *destination* is filled in with the information previously collected. This scenario is executed once the user reaches the page *booking.com* (either by following a link or entering a new URL). Notice that the scenario can be instantiated, because the information needed is available into the *Pocket*. For form filling cases, the adaptation could be automatic when the adaptation is developed particularly for an application (in this case for Booking.com) or even by using other tools like carbon [1] in order to automatically fill forms in any Web site. This use of concern information improves the user experience by allowing him to “transport” critical data among Web applications and use these data to adapt them.

Data Collection in Expedia.com

The screenshot shows a flight search result for a roundtrip from Buenos Aires to Paris. The departure date is Fri 11-Feb-11 and the arrival date is Fri 25-Feb-11. The total price is £1,280.70. A 'PopUp Message' is overlaid on the page, asking if the user wants to book hotels and providing a link to OpenBooking.com.

1 Review the flight details
Fri 11-Feb-11
Buenos Aires (EZE) to Paris (CDG)
Depart 16:10 Arrive 11:00 +1 day
Economy/Coach Class , 77W
Total distance: * (mileage not available for all flights)
Fri 25-Feb-11
Paris (CDG) to Buenos Aires (EZE)
Depart 23:20 Arrive 06:50

depart Date
destination
arrive Date

Form filling in Booking.com with data collected previously

The screenshot shows the Booking.com 'Search Hotels' page. The 'Destination' field is populated with 'Paris'. The 'Check-in date' is set to 'Fri 11 February '11' and the 'Check-out date' is set to 'Fri 25 February '11'.

BOOKING.COM
online hotel reservations
English (UK) U.S. English (US)
home

Search Hotels
Destination
Check-in date
Check-out date

Fig 9. a. Information extraction from *expedia.com*

Fig 9. b. Form filling in *booking.com* with information collected in previous Web sites



Fig. 10. a. Information in pocket used into a GoogleMaps scenario



Fig. 10. b. Information in pocket used into a Flickr scenario

For example in Figure 10.a we show how, when the user arrives to Google Maps, the information in the pocket can be used automatically to create Google Maps links in the left bar. On the other hand, the same information can be used in another scenario if the user opens Flickr.com as shown in Figure 10.b where the points of interest are offered as Flickr's tags. In this adaptation, the scenario engineer has used a framework tool (the floating box is a PopUpMessage) for a message suggesting a simple adaptation.

5 Evaluation of the Approach

To validate our approach and actual usage of the tools, we have conducted a usability study with end-users. The goal of this evaluation was to investigate if client-side adaptation is usable for solving common tasks whilst navigating the web. The adaptations investigated in this study explored the following framework components: *Highlight* for changing color of important information, *WikiLinkConversion* for creating new links to Wikipedia, *DataCollector* for recording information for later usage, and *CopyIntoInput* for automating filling in forms with dates previously collected by the user.

The study was run with 11 participants (6 males and 5 females, aged from 23 to 46 years old). All participants were experienced Web users (i.e. > 5 years using the Web) that spend a significant time browsing the Web as part of their daily activities (in average 4,1 hours of navigation on the Web per day, SD=2,4 h). We have focused on experienced users because we assume that they are more likely to formulate special needs for adapting Web pages than novices with the Web. Participants were asked to fill out a pre-questionnaire; following they were introduced to the system and asked to conduct five tasks at their workplace, followed by a final interview and a System Usability Scale questionnaire (i.e. SUS, [3]). The SUS has been used as a complement to user observation, as it is widely used in comparative usability assessments in industry. The five tasks were related to investigate the working hypothesis, on how usable our approach is for solving common tasks whilst navigating Web sites. All tasks were related to the following problem: the goal is to plan a trip to Paris to visit an exposition, which includes collecting information such as dates and location

of the exposition and booking a hotel; for that purpose users should visit different Web sites and use our tools to perform client-side adaption on the page visited. In average, users spent 37 minutes to complete the test. Usability was measure in terms of time to accomplish tasks, number of tasks performed successfully, and user satisfaction (via a questionnaire).

The results show that, generally, participants appreciate the concept of client-side adaptation and the tool support. In the pre-questionnaire, when asked if they would like to modify the Web pages they visit, 2 of 11 participants said no because “it could be very time consuming”. Notwithstanding, all participants said that our tools for client-side adaptation are useful and that they are willing to use them in the future. Adaption across different Web site was described as “natural” by 7 participants and a “real need” by 5 of them. The component *DataCollector* was the most successfully applied by all participants; it was considered very useful and a “good substitute for post-its”. However, success rate varied according to the augmenter employed: *CopyIntoInput* was considered very easy to use by participants and employed successfully by 10 of them (90,9%). The augmenter *highlight* (72% of success rate, 8 participants) was considered easy to use but 5 users blamed it because they could only apply it to the exact word previously selected, and users cannot choose the color and/or the police used to highlight different pieces of information. Participants were very impressed by the augmenter allowing links to Wikipedia from concepts (the *WikiLinkConversion*); despite the fact that it was considered extremely useful, the success rate with this augmenter was the lower in the study, 18%, due to two main issues: the fact that links can only be created from typed information and lack of visual feedback (i.e. an icon) indicating where that action was possible. Nine participants (81,8%) said that using the augmenters improved their performance with tasks, one user said it could be faster without the augmenters and the other one didn’t see any difference. This user perception has been confirmed by the time recorded during task execution using augmenters *WikiLinkConversion* and *CopyIntoInput*.

This study also revealed some usability problems that motivate further development in the tool. For example, users requested to have a visual indicator allowing them to distinguish where augmenters have been applied (ex. links on the Web site x links created with the augmenter *WikiLinkConversion*). Users intuitively tried to activate some of the augmenters using *Drag & Drop* which is an indicator for further research of more natural interaction with *augmenters*. The most frequent suggestions for new augmenters include “automatic filling forms”, “create links to other Web sites than Wikipedia”, and “automatic highlight at the Web page of information previously collected”. This positive analysis is confirmed by a SUS score of 84,9 points ($SD = 5,5$), which is a good indicator of general usability of the system.

6 Conclusions and Future Work

In this work we have presented a novel approach for client-side adaptation which takes into account the tasks that users perform while navigating the Web. We aim to support complex concern sensitive adaptations in the client-side in order to improve the users’ experience. We have developed a support framework which can be extended with two kinds of adaptations: atomic augmenters (realizing simple adaptation actions) and scenarios which comprise the use of different augmenters on somewhat predefined Web pages. These adaptations can be executed either manually, e.g.

when the user triggers an adaptation action explicitly, or automatically when some scenario is instantiated. Being built on solid engineering principles, the framework can be extended and/or used both by end-users or developers (e.g. by developing JavaScript code). In comparison with the usual client-side adaptations, we provide a flexible mechanism to integrate information while users navigate the web, instead of “just” providing tools to statically adapt Web sites. Our approach is based in two main types of developers interventions: the first one (augmenters) supports generic scripts with specific adaptation goals to be applied over any Web page, and the second one (scenarios) can be used when the goal is to support users tasks among several Web sites. We have performed a small but meaningful evaluation with end-users with excellent results.

We are working in several directions to improve the approach. The first one is to improve the development process and tools for developers using the framework. Although we have defined guidelines for both augmenters and scenarios development, these must still be written in a quite similar way to bare JavaScript programming. Our goal is to raise the abstraction level for developers by creating a domain specific language that will simplify the specification of both augmenters and scenarios; this will let users without JavaScript knowledge to develop adaptations easily.

Besides that, and as indicated in Section 5 we have detected usability problems in some of our tools when users are trying to adapt the Web sites or even while they are collecting data. In this sense we are developing not only new tools but also tuning the existing ones and performing new evaluations with them.

References

1. Araujo, S., Gao, Q., Leonardi, E., Houben, G.-J.: Carbon: Domain-Independent Automatic Web Form Filling. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 292–306. Springer, Heidelberg (2010)
2. Bouvin, N.O.: Unifying Strategies for Web Augmentation. In: Proc. of the 10th ACM Conference on Hypertext and Hypermedia (1999)
3. Brooke, J.: SUS: a ‘quick and dirty’ usability scale. In: Usability Evaluation in Industry. Taylor and Francis, London (1996)
4. Brusilovsky, P.: Adaptive Navigation Support. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) Adaptive Web 2007. LNCS, vol. 4321, pp. 263–290. Springer, Heidelberg (2007)
5. Daniel, F., Casati, F., Soi, S., Fox, J., Zancarli, D., Shan, M.: Hosted Universal Integration on the Web: The mashArt Platform. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 647–648. Springer, Heidelberg (2009)
6. Diaz, O., Arellano, C., Iturrioz, J.: Layman tuning of websites: facing change resilience. In: Proc. of WWW2008 Conference, Beijing, pp. 1127–1128 (2008)
7. Díaz, O., Arellano, C., Iturrioz, J.: Interfaces for Scripting: Making Greasemonkey Scripts Resilient to Website Upgrades. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 233–247. Springer, Heidelberg (2010)
8. Firmenich, S., Rossi, G., Urbieto, M., Gordillo, S., Challiol, C., Nanard, J., Nanard, M., Araujo, J.: Engineering Concern-Sensitive Navigation Structures. Concepts, tools and examples. In: JWE 2010, pp. 157–185 (2010)
9. Greasemonkey,
<http://www.greasespot.net/> (last visit on February 11, 2011)

10. Han, H., Tokuda, T.: A Method for Integration of Web Applications Based on Information Extraction. In: Proceeding of ICWE, New York, pp. 189–195. Springer, Heidelberg (2008)
11. Meusel, M., Czarnecki, K., Köpf, W.: A Model for Structuring User Documentation of Object-Oriented Frameworks Using Patterns and Hypertext. In: Proceedings of ECOOP 1997, pp. 496–510 (1997)
12. Miller, C.S., Remington, R.W.: Modeling an Opportunistic Strategy for Information Navigation. In: Proc. Of 23rd Conference of the Cognitive Science Society, 2001, pp. 639–644 (2001)
13. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding Mashup Development. IEEE Internet Computing 12, 44–52 (2008)
14. Wong, J., Hong, J.I.: Making Mashups wit Marmite: Towards End-User Programming for the Web. ACM, City (2007)

Instantiating Web Quality Models in a Purposeful Way

Philip Lew¹ and Luis Olsina²

¹ School of Software, Beihang University, China

² GIDIS_Web, Engineering School, Universidad Nacional de La Pampa, Argentina
philiplew@gmail.com, olsinal@ing.unlpam.edu.ar

Abstract. Web applications and their quality evaluation has been the subject of abundant research. However, models have been used mostly for the purpose of understanding, rather than improving. In this work, we propose utilizing a quality modeling framework to instantiate quality models with the specific purpose to not only to understand the current situation of an entity, but also to improve it. Our approach instantiates models for both external quality and quality in use, resulting in a requirements tree for both followed by evaluation and then combined with a mechanism to develop relationships between them. Hence, improving is driven by understanding these relationships, namely, ‘depends on’, and ‘influences’ in alignment with the ISO 25010 quality life cycle model. This is illustrated with a case study showing the underlying strategy from model instantiation to application improvement.

Keywords: Quality improvement, quality in use, actual usability, external quality, SIQinU strategy.

1 Introduction

Today’s web-based applications (WebApps) containing complex business logic and sometimes critical to operating the business, are now requiring increased focus on understanding and improving their quality. One of the first steps to evaluate quality is to define nonfunctional requirements usually through quality models. The ISO 25010 standard [11] describes one such model for general usage in specifying and evaluating software quality requirements. However, ISO 25010 is intended as a general guideline to be adapted based on a specific information need and context, i.e. for evaluating WebApps. In addition, some of ISO model concepts, while founded strongly in theory, are difficult to realize in a real situation particularly when it comes to measuring and evaluating quality in use (QinU).

But the main goal in evaluating software quality is to ultimately improve. However, independent from models, conducting evaluations in real situations particularly for QinU is difficult to realize. Therefore, a key issue is to relate QinU evaluation results to properties intrinsic to the WebApp itself in order to make improvements. In modeling terms, QinU characteristics and attributes need to be related to external quality (EQ) characteristics and attributes. That is to say, does the software’s new (and improved) version have a positive impact on its QinU?

To answer this question for WebApps, we began by first proposing to augment the ISO 25010 standard through using the 2Q2U (*Quality, Quality in use, actual Usability*)

(and *User experience*) modeling framework [12] to include *information quality* as a characteristic of internal/EQ because this is a critical characteristic of WebApps. We further proposed to include *learnability in use* as a characteristic of *usability in use* to account for the learning process and the importance of context of use during learning. 2Q2U relies on the ISO 25010 premise that the relationships ‘depends on’ and ‘influences’ exist between EQ and QinU. Using this premise, we further utilize 2Q2U to instantiate models for both EQ and QinU specifically for the purpose of improving the QinU of a WebApp. However, the ISO 25010 premise that QinU depends on EQ and in turn EQ influences QinU is very general, not specifically explored, and there is no description on implementing or using these relationships for purposeful evaluations.

For this reason, we devised SIQinU (*Strategy for Improving Quality in Use*), a strategy for improving quality that also uncovers these relationships in a systematic way. Starting with QinU, we design specific tasks and context of use, and through identifying problems in QinU, we determine EQ attributes that could be related to these QinU weakly performing indicators. Then, after deriving EQ attributes related to the QinU problems, we evaluate EQ and derive a benchmark to be used as a basis to make improvements. Once improvement recommendations are made based on poorly performing EQ measurements (related to the poorly performing QinU indicators), a new version of the WebApp is completed and evaluated again for its EQ to establish a delta from the initial benchmark. Then we re-evaluate QinU to determine the improvements resulting in QinU from the improvements made at the EQ level, thus leading to a cyclic strategy for improvement and development of relationships. These relationships between EQ and QinU are not just for understanding but developed with the primary objective of improving the application with respect to poorly performing QinU attributes. Thus, information regarding ‘depends on’ and ‘influences’, as depicted in the ISO 25010 quality lifecycle model, are extracted in the process of improving the WebApp. Armed with these relationships, designers can then design/improve software at the EQ level knowing the impact on QinU. Through employing SIQinU, this work, in particular, focuses on improving the *actual usability* of WebApps from an end user viewpoint when executing real tasks in a real context.

Aligning with the ISO 25010 models on the quality life cycle, SIQinU combined with 2Q2U utilizes the ISO premise that if quality can be improved at the EQ point of view, this influences and most likely also improves quality from the QinU viewpoint. The proposed SIQinU strategy utilizes the 2Q2U framework for modeling requirements, non-intrusively collects user behavior data, and provides an integrated means to use quality models in a real context to evaluate EQ and QinU for WebApps with a primary objective of improvement through an evaluation process [2] and methods that are consistent and repeatable.

Consistent and repeatable is a paramount characteristic of SIQinU in order to iteratively improve a WebApp. This is gained primarily through a non-functional requirements ontological component of the C-INCAMI (*Contextual-Information Need, Concept model, Attribute, Metric and Indicator*) framework [15], which enables us to instantiate a project (our project consisted of improving a concrete actual WebApp) that includes defining the information need, user viewpoint, entity, and so forth. Ultimately, the contributions of this research are:

- A procedure to concretely instantiate quality models based on 2Q2U and C-INCAMI for both QinU and EQ for the purpose of not only understanding but also improving a WebApp.
- A concrete strategy, SIQinU, combined with a well-defined and established process to guide the improvement with an illustration of the improvement results through a case study.
- An exploration of the relationships outlined in the 25010 quality lifecycle model, namely influences and depends, gained through the process of improvement in both EQ and QinU instantiated views.

Following this introduction, Section 2 reviews recent related work and delineates opportunities for improvements which are the motivation for this research. Section 3 demonstrates our procedure for using 2Q2U to purposefully instantiate models for improving a WebApp, which can then be utilized by SIQinU to improve a concrete WebApp. In Section 4, we use the instantiated models with the strategy in a case study in the context of evaluating EQ and QinU for a WebApp with the goal of improvement while deriving possible relationships between EQ and QinU. Section 5 draws our main conclusions and outlines future work.

2 Related Work and Motivation

In this paper, we instantiate quality models for the purpose of improvement and then combine these instantiations with a strategy to carry out evaluations to ultimately accomplish improvement. As such, based on our examination of existing research, there has been progress in the individual elements such as modeling and evaluation, but limited focus on using a strategy and tailored models for the purpose of improvement considering the QinU/EQ/QinU cycle. Regarding improvement strategies by evaluation for WebApps, in [16] authors present an approach for incremental EQ improvement. Their work uses the results from EQ evaluation to make changes and improvements in a WebApp through WMR (*Web Model Refactoring*) but the EQ requirements were not mapped from real QinU problems and also lacks a strategy for continual improvement. In [7] authors propose a systematic approach to specify, measure and evaluate QinU, but the outcomes were only used to understand the current QinU satisfaction level for an e-learning WebApp, without proposing any improvement strategy. Conversely, the GQM⁺Strategies approach [1] is an integrated strategy for defining and satisfying measurement goals, but does not give explicit steps to guide the evaluation and improvement. Lastly, in [8] authors present a generic usability evaluation process which can be instantiated into any model-driven web development process, but no improvement strategy is discussed.

Regarding the derivation of EQ characteristics and attributes from QinU requirements and problems, there is a related initiative [13], which focuses on employing a Bayesian method in order to find out influences of EQ characteristics on QinU characteristics. However, this work has limited practical benefit because the derivation is theoretical rather than using a real context of use. Moreover, there is no integrated improvement strategy, but rather just a derivation technique.

There are many works aimed at increasing WebApp quality by establishing automated procedures for product improvement during development stages. Meanwhile others use user evaluation at early lifecycle stages. As an example of the former, [5], design patterns that influence quality are included at the conceptual modeling phase and achieved into the WebApp code by means of model transformations. The latter is illustrated by the TRUMP methodology [4], which defines a set of methods to apply to each of the phases and processes described in [10]. This methodology allows evaluating (testing) by a set of users, with an early version of the application, identifying usability problems and then establishing usability requirements for improving the product. Thus in the end, the product is re-evaluated by users to observe whether usability goals were achieved, but there is no strategy for continual improvement through connecting EQ and QinU.

Summarizing the existing research, there lacks attention for models and their instantiation for the goal of improvement. Given that, this work aims to use the 2Q2U framework [12] to instantiate quality models for both EQ and QinU, followed by a strategy that uses these models and purposely performs evaluations with the end goal in mind: improving the WebApp. And through the improvement cycle, potential relationships are drawn between EQ and QinU which are useful not only for the improvement of the WebApp under study, but also possibly applicable to other WebApps leading to further research areas.

3 Instantiating Quality Models in a Purposeful Way

Our view for this research is that understanding is the means and improvement is the ultimate goal. With modeling and evaluation as steps toward improvement, models must be instantiated with this in mind. So, starting with modeling, we use the 2Q2U modeling framework to instantiate a concept model for both QinU and EQ. With 2Q2U, we include *learnability in use* as a sub-characteristic of QinU and *information quality* as characteristic of EQ. We also define two new concepts *actual user experience* and *actual usability* where the latter encompasses the ‘do goals’ of the user, as defined by ISO 25010 usability, with the exception of the *satisfaction* sub-characteristic related to ‘be goals’ and modeled as a characteristic of actual user experience [3, 9]. Details on these definitions and relationships can be found in [12].

To use the instantiated models for measurement and evaluation (M&E) in a consistent and repeatable way, we utilize the C-INCAMI [15] framework. The C-INCAMI framework defines all of the concepts and relationships needed to design and implement M&E processes. C-INCAMI’s approach is designed to satisfy a specific information need in a given context defining concepts and relationships that are used along all the M&E activities lending to consistent analysis and results. The framework has six components: i) *M&E project definition*, ii) *Nonfunctional requirements specification*, iii) *Context specification*, iv) *Measurement design and implementation*, v) *Evaluation design and implementation*, and vi) *Analysis and recommendation specification*. Of particular use for this research in instantiating quality models for the purposes of improvement is C-INCAMI’s Nonfunctional Requirements Specification (NFRS), and Context Specification component, shown in Figure 1.

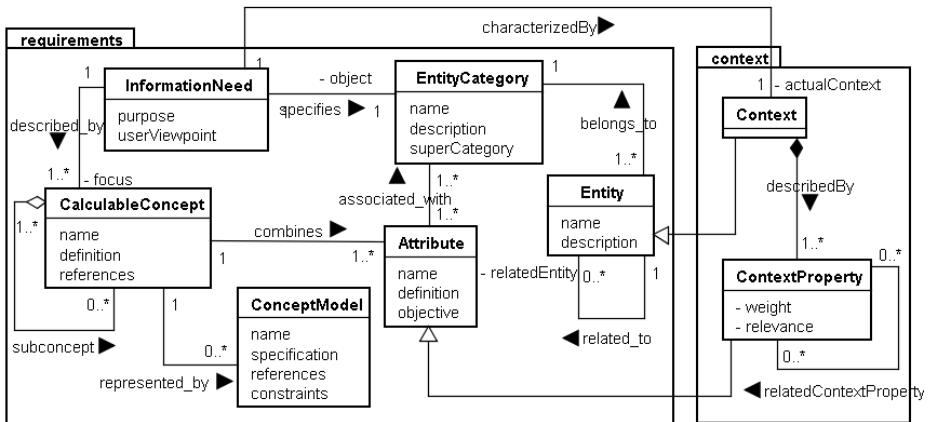


Fig. 1. C-INCAMI Nonfunctional requirements and context specification components

The NFRS specifies the *Information Need* of any M&E project; that is, the *purpose* (e.g. “understand”, “predict”, “improve”, etc.) and the *user viewpoint* (e.g. “developer”, “final user”, etc.). In turn, it focuses on a *Calculable Concept* and specifies the *Entity Category* to evaluate –e.g. a resource, process, product, etc.–, by means of a concrete *Entity* –e.g., the amazon.com shopping basket. A calculable concept can be defined as an abstract relationship between attributes of an entity and a given information need. In our case, the purpose is to improve, a WebApp, regarding its EQ and QinU, and its actual usability as the calculable concept. This can be represented by a *Concept Model* where the leaves of an instantiated model (e.g. a requirements tree) are *Attributes* associated with an *Entity*.

Context Specification delineates the state of the situation of the entity to be assessed with regard to the information need. *Context*, a special kind of *Entity* in which related relevant entities are involved, can be quantified through its related entities that may be resources –as a network infrastructure, a working team, life-cycle process-, the organization or the project itself, among others. In our case, the context is particularly important regarding QinU requirements as instantiation of requirements must be done consistently in the same context.

For this paper, regarding SIQinU, we utilize 2Q2U to create the models combined with the NFRS and Context Specification components of C-INCAMI to conduct the evaluations as an integral part of the strategy. Using the C-INCAMI terminology and framework as shown in Figure 1, the information need is to understand and improve, while the entity is a particular WebApp who’s category is Web Application. In particular, later in the case study illustration, we exhibit an actual WebApp as well. The concept model used is the instantiated 2Q2U framework shown in Figure 3 while the calculable concepts are *usability in use*, *information quality*, and *operability*. In particular, our objective is to understand both the current and future evaluations of the entity in order to determine improvement after changes were made to the current version. Employing the 2Q2U framework, we purposely instantiate it with operability and information quality as EQ characteristics to be related to the QinU characteristic, actual usability combined with sub-characteristics: *efficiency in use*, *effectiveness in use*, and *learnability in use* as shown in Figure 3.

Table 1. SIQinU Phases, activities and work products

Phase (Ph.)	Description/Activities	Work Products
<i>Ph. I</i> Specify Requirements and Evaluation Criteria for QinU	Taking into account the recorded data of the WebApp's usage, we re-engineer QinU requirements. This embraces designing tasks, defining user type, specifying usage context and characteristics, particularly, for <i>actual usability</i> as defined in [12]. Activities include: i) Establish Information Need; ii) Specify Project Context; iii) Design Tasks; iv) Select QinU Concept Model; v) Design QinU Measurement and Evaluation; vi) Design Preliminary Analysis	- Information Need and Context specification - QinU NFRS tree - QinU Metrics and Indicators specification - Task/sub-tasks specification
<i>Ph. II</i> Perform QinU Evaluation and Conduct Preliminary Analysis	As per Ph. I, data is collected purposely targeting QinU attributes for improvement. Depending on the WebApp's ability to collect the data, we also collect the date/time the data is gathered, errors, task and sub-task completion and accuracy, etc. It includes: i) Collect and parse data pertaining to tasks with their sub-tasks; ii) Quantify QinU Attributes; iii) Calculate QinU Indicators; iv) Conduct preliminary analysis.	- Measure and indicator values for QinU - QinU preliminary analysis report
<i>Ph. III</i> Derive/Specify Requirements and Evaluation Criteria for EQ	Based on Ph. I and II, we derive EQ requirements, i.e. characteristics and attributes, with their metrics and indicators in order to understand the current WebApp's quality. In our case study, a focus on <i>operability</i> and <i>information quality</i> was used to determine possible effects and specifically improve <i>actual usability</i> . Activities include: i) Determine EQ Concept Model; ii) Design EQ Measurement; iii) Design EQ Evaluation	- EQ NFRS tree - EQ Metrics and Indicators specification
<i>Ph. IV</i> Perform EQ Evaluation and Analysis	Activities include: i) Quantify EQ Attributes; ii) Calculate EQ Indicators; iii) Conduct an EQ analysis and identify parts of the WebApp that need improvement.	- Measure and indicator values for EQ - EQ Analysis report
<i>Ph. V</i> Recommend, Perform Improvement Actions, and Re-evaluate EQ	Using the EQ attributes that require improvement, we make improvement recommendations for modifying the WebApp, i.e. version 1 to 1.1. Activities include: i) Recommend improvement actions; ii) Design Improvement Actions; iii) Perform Improvement Actions; iv) Evaluate Improvement Gain to note improvement from benchmark in Ph. IV	- EQ Recommendations report - Improvement plan - New app version - EQ Analysis report
<i>Ph. VI</i> Re-evaluate QinU and Analyze Improvement Actions	Once the new version has been used by real users, we evaluate QinU again to determine the influence of what was improved for the WebApp's EQ on QinU. This provides insight to further develop the <i>depends</i> and <i>influences</i> relationships. Activities include: i) Evaluate QinU again to determine level of improvement from Ph.II; ii) Conduct Improvement Action analysis; iii) Develop depends and influences relationships between EQ improvements and QinU.	- Measure and indicator values for QinU - QinU Improvement analysis report - EQ/QinU attribute relationship table

Note that we could have selected other characteristics for both EQ and QinU, but we purposely instantiated these characteristics as per our information need via the C-INCAMI NFRS component. Later in Section 4, we fully derive the requirements tree for each of the sub-characteristics at a more detailed level in illustrating the case study.

The proposed SIQinU strategy utilizes the 2Q2U framework for quality models and the C-INCAMI framework for instantiating M&E projects in a purposeful way. SIQinU, non-intrusively collects user behavior data from a real context of use, and provides an integrated means to evaluate QinU and EQ for WebApps with a primary objective of improvement through an evaluation process [2] and methods that are consistent and repeatable.

SIQinU collects user behavior data from log files [6] that were derived through for example adding snippets of code in a real WebApp-in-use that allow recording that data, with an aim to derive nonfunctional requirements measures and indicators for QinU, thus leading us to understand the current QinU satisfaction levels met. Then, by performing a preliminary analysis we derive EQ requirements that can affect QinU, and propose recommendations for improvements. After performing the changes on the WebApp, and after conducting studies with the same user group in the same daily environment (context) with the new version, an assessment of the improvement gain can be gauged. With this knowledge of improvement, we can then extract relationships in a cyclic manner by isolating changes in EQ attributes that affected QinU attributes in a positive way. Thereby, each iteration between QinU and EQ can result in continued improvement.

SIQinU's phases are illustrated in Figure 2, combined with Table 1 which provides a brief description with Phase (Ph.) reference numbers, activities and work products.

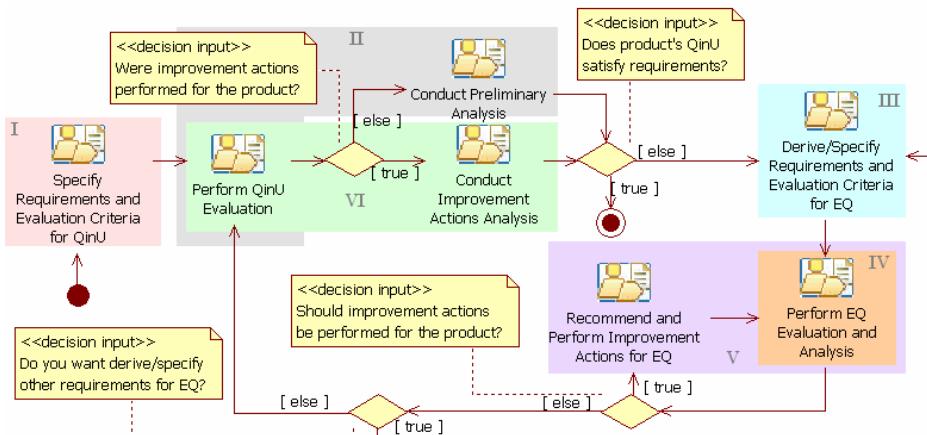


Fig. 2. Process overview for understanding and improving quality in use (SIQinU)

Ultimately, in the process of using SIQinU, we are able to gain insight regarding the *depends on* and *influences* relationships for the particular 2Q2U instantiated models, and their characteristics and attributes driven by our purpose to improve. In addition, we can continue to iterate the SIQinU improvement cycle to gain further insight and granularity adding a temporal component for later study.

4 A Quality Improvement Lifecycle Using SIQinU: A Case Study

This section illustrates the quality improvement lifecycle via SIQinU using excerpts of a case study conducted in mid-2010. It examined JIRA (www.atlassian.com), a defect reporting WebApp in commercial use in over 14,500 organizations in 122 countries. JIRA's most common task, *Entering a new defect*, was evaluated in order to provide the most benefit, since entering a new defect represents a large percentage of the total usage of the application. We studied approximately 50 beginner users in a real work environment in their daily routine of reporting defects in a software testing department –in a real company specializing in software quality and testing. Although there are other user categories such as test managers, QA managers, and administrators, testers is the predominant user type, so we chose beginner testers as our user viewpoint. Next, we discuss some aspects of the above phases; mainly those aimed at showing models instantiation, EQ and QinU improvement gains and potential uncovered relationships.

In Ph.I, we start by using 2Q2U to purposefully instantiate a QinU model from the *actual usability* standpoint. We first want to *understand* the current situation of the entity (JIRA v.1) from the *beginner user* viewpoint performing the above mentioned task. To do this, we design the QinU requirements as shown in the right part of Figure 3, resulting in the requirements tree shown in the column 1 of Table 2.

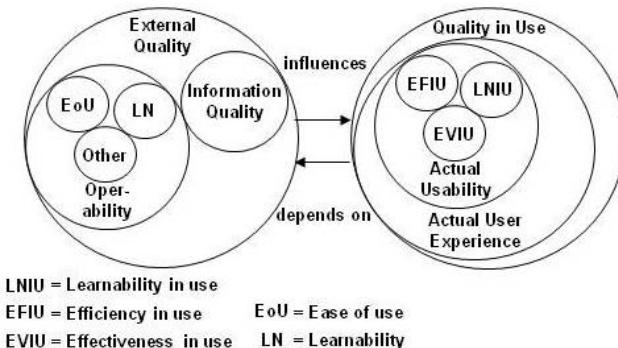


Fig. 3. 2Q2U model instantiation with some EQ and QinU characteristics shown

Using requirements from Ph. I of our instantiated model, we conduct the QinU evaluation as Ph. II of SIQinU. This evaluation is done for each attribute specified in Ph. I, and results in a global evaluation of 53.3% as shown in Table 2. Note that in *designing the QinU evaluation* (Ph. I activity shown in Table 1) for each attribute of the requirements tree we specified an elementary indicator with three acceptability ranges in the percentage scale, namely: a value within 70-90 (a marginal –gray-range) indicates a need for improvement actions; a value within 0-70 (an unsatisfactory –dark gray- range) means changes must take place with high priority; a score within 90-100 indicates a satisfactory level –light gray- for the analyzed attribute.

Table 2. QinU requirements tree and evaluation results of JIRA v.1 (before) and JIRA v.1.1 (after modifications). EI stands for Elementary Indicator; P/GI for Partial/Global Indicator.

Characteristics and Attributes	JIRA v.1		JIRA v.1.1	
	EI	P/GI	EI	P/GI
1. Actual Usability		53.3%		67.0%
1.1. Effectiveness in use		73.2%		86.7%
1.1.1. Sub-Task Correctness	86.4%		91.9%	
1.1.2. Sub-Task Completeness	87.9%		95.5%	
1.1.3. Task Successfulness	45.5%		72.7%	
1.2. Efficiency in use		29.3%		42.8%
1.2.1. Sub-Task Correctness Efficiency	37.4%		44.3%	
1.2.2. Sub-Task Completeness Efficiency	37.5%		47.3%	
1.2.3. Task Successfulness Efficiency	13.1%		36.8%	
1.3. Learnability in use		57.3%		71.6%
1.3.1. Sub-Task Correctness Learnability	78.8%		75.1%	
1.3.2. Sub-Task Completeness Learnability	26.4%		77.3%	
1.3.3. Task Successfulness Learnability	66.7%		62.5%	

Based on the evaluation, using the defined indicators, we are able to determine which attributes had low performance or problems, e.g., *sub-task completeness learnability* which had an unsatisfactory rating of 26.4%. We can rank them in terms of priority by lowest performing at the top, or depending on our requirements, which may weight other attributes more heavily, do a more complex priority ranking accordingly.

SIQinU in this case study using JIRA was implemented in a non-intrusive way through interpretation of log files and automated application and calculation of indicators thereby resulting in list of problem areas (those attributes that rated unsatisfactory). Note that other complementary techniques could be used as traditional observational techniques [14] to understand user problems while interacting with the selected tasks and to help deriving EQ attributes. However a trade-off between costs and benefits should be carefully considered.

Results from Ph. II (from the *conduct preliminary analysis* activity) are then used to derive EQ requirements for Ph. III. Examining the relevant QinU problems associated with the task/sub-tasks and screens in the WebApp, and using 2Q2U we (as experts) derived the EQ model which included *operability*, and *information quality* characteristics. This results in deriving a requirements tree of EQ attributes for those particular QinU attributes that had problems (see the left hand column of Table 3).

Note that in the first 3 phases, in going from QinU requirements to QinU problems and then eventually to EQ requirements as just discussed, the quality lifecycle model is thereby deployed and the “depends on” relationship (see Fig. 3) is expanded to include attributes of our instantiated models. Then, in Ph. IV, we evaluate JIRA v.1 by inspection using the EQ instantiated requirements tree whereby each attribute and metric is calculated and evaluated based on the defined indicators. Table 3 shows the results of the EQ evaluation in columns 2-3.

Examining the evaluation and the indicators which show that some attributes perform lowly, we then make recommendations for improvement. These

Table 3. EQ derived requirements tree and evaluation results of JIRA v.1 (before) and JIRA v.1.1 after implementing improvements on the concrete WebApp

Characteristics and Attributes	JIRA v.1		JIRA v.1.1	
	EI	P/G I	EI	P/G I
External Quality		38%		74%
1. Operability		30%		60%
1.1. Learnability		26%		59%
1.1.1. Feedback suitability		38%		38%
1.1.1.1. Navigability feedback completeness	33%		33%	
1.1.1.2. Task progress feedback appropriateness	30%		30%	
1.1.1.3. Entry form feedback awareness	50%		50%	
1.1.2. Helpfulness		15%		80%
1.1.2.1. Context-sensitive help availability	20%		80%	
1.1.2.2. Help completeness	10%		80%	
1.2. Ease of use		34%		61%
1.2.1. Controllability		80%		80%
1.2.1.1. Permanence of main controls	60%		60%	
1.2.1.2. Stability of main controls	100%		100%	
...

recommendations are given to the M&E project sponsor whereupon evaluators/developers may choose a variety of methods to make the changes depending on the resources they have available. This could range from a complete restructuring of code to simple menu configuration changes.

In Ph.V, after the evaluation-driven improvements are made on the WebApp (now named JIRA v.1.1), we re-evaluate EQ using the same EQ requirements and note the improvement gain. More than likely, some but not all of the improvements will have been made due to resource availability and difficulty to make the recommended changes. This second evaluation leads to a linkage between specific improvements made to the WebApp and the affected EQ attribute which was hopefully improved.

With the improved application, we re-evaluated QinU using the same requirements and note the changes. Hopefully, since there were improvements from the EQ standpoint, there will be improvements in the QinU of the WebApp, thus enabling us to begin to understand the relationships between the EQ and QinU attributes instantiated in our model development. Table 2 shows the QinU evaluation for the improved WebApp version of JIRA v.1.1 showing noticeable improvement in many attributes and an improvement from 53.3% to 67% for the global indicator for *actual usability*. Comparing each attribute in more detail, Table 4, shows all attributes noted improvement with the exception of *Task Successfulness Learnability* and *Sub-task Correctness Learnability*. However, their negative change was small compared to the positive changes in the other attributes resulting in an overall average change of attributes evaluation of 13.7%. While the indicators show that most of the attributes in JIRA v.1.1 still need some or significant improvement, there has been notable improvement from JIRA v.1.

Table 4. Actual usability evaluated attributes for JIRA v.1 and v.1.1 with improvements gain

Attributes	JIRA v.1	JIRA v.1.1	Change (+ is improvement)
1.1.1. Sub-Task Correctness	86.4%	91.9%	5.5%
1.1.2. Sub-Task Completeness	87.9%	95.5%	7.6%
1.1.3. Task Successfulness	45.5%	72.7%	27.2%
1.2.1. Sub-Task Correctness Efficiency	37.4%	44.3%	6.9%
1.2.2. Sub-Task Completeness Efficiency	37.5%	47.3%	9.8%
1.2.3. Task Successfulness Efficiency	13.1%	36.8%	23.7%
1.3.1. Sub-Task Correctness Learnability	78.8%	75.1%	-3.7%
1.3.2. Sub-Task Completeness Learnability	26.4%	77.3%	50.9%
1.3.3. Task Successfulness Learnability	66.7%	62.5%	-4.2%
Average Change			13.7%

The next activity of this final phase (recall Table 1) for this cycle of SIQinU involves examining possible relationships between EQ and QinU attributes based on our two versions of JIRA. Table 7 shows an excerpt of the relationships derived. Relationships derived can then be used as input to the next iteration of SIQinU whereby those identified ‘depends on’ and ‘influences’ can then be purposefully instantiated in Ph. I of subsequent SIQinU cycles.

Note that Phases IV to VI, in going from EQ requirements to QinU improvement, the quality lifecycle model is thereby deployed and the ‘influences’ relationship is again exemplified to include specific attributes of our instantiated models with possible degrees of relationship, although not statistically done at this point. Further statistical studies could be done for instance, by isolating one particular attribute in EQ and QinU and going through the SIQinU cycle of improvement. In illustrating the SIQinU improvement cycle, namely QinU/EQ/QinU and instantiating with the purpose of understanding and improving, we therefore can explore relationships between EQ and QinU not only for the WebApp under study, but for WebApps in general and use the strategy to continue improvement in a consistent way.

Based on this, with our defined task of *Entering a new defect*, Table 5 lists out the *actual usability* attributes ranked in terms of improvement gain. The higher levels of improvement possibly indicate that changes made at the EQ level had a greater influence.

Table 5. Analysis of actual usability attributes ranked by improvement

Ranking by Evaluation	change (+ is improvement)
1.3.2. Sub-Task Completeness Learnability	50.9 %
1.1.3. Task Successfulness	27.3 %
1.2.3. Task Successfulness Efficiency	23.8 %
1.2.2. Sub-Task Completeness Efficiency	9.8 %
1.1.2. Sub-Task Completeness	7.6 %
1.2.1. Sub-Task Correctness Efficiency	6.8 %
1.1.1. Sub-Task Correctness	5.6 %
1.3.1. Sub-Task Correctness Learnability	-3.7%
1.3.3. Task Successfulness Learnability	-4.2%

Table 6. Changes made from JIRA v.1 to v.1.1 (excerpt) based on recommendations

Related QinU Attribute	QinU Attribute Improvement	EQ Related Attribute	EQ Attribute improvement	Improvement Recommendation
Learnability in use: Sub-task completeness learnability	50.9%	2.1.1 Information quality.InfoSuitability.Consistency	50.0%	Change fields to have most important information before the details
Learnability in use: Sub-task completeness learnability	50.9%	1.1.2.2 Learnability.Helpfulness.HelpCompleteness	70.0%	Add context sensitive help

Table 7. Relationships developed during the JIRA case study for EQ and QinU attributes

Related QinU Attribute	Related EQ Attribute
Learnability in use: Sub-task completeness learnability	1.1.2.2 Learnability.Helpfulness.HelpCompleteness 2.1.1 Information quality.InfoSuitability.Consistency
Effectiveness in use: Task Successfulness	1.2.1.2 Ease of use.Controlability.StabilityofMainControls 1.1.1.2 Learnability.Feedback Suitability.TaskProgressFeedbackAppropriateness 1.1.1.3 Learnability.Feedback Suitability.EntryFormFeedbackAwareness 1.1.2.1 Learnability.Helpfulness.Context-sensitiveHelpAvailability
Efficiency in use: Sub-task completeness efficiency	1.2.3.1 Ease of use.Data Entry Ease.Defaults 1.2.3.2 Ease of use.DataEntryEase.MandatoryEntry 1.2.3.3 EaseofUse.DataEntryEase.ControlAppropriateness 2.1.2.1 Information quality.InfoSuitability.InfoCoverage.Appropriateness
Effectiveness in use: Sub-task completeness	2.1.1.1 Information quality.InfoSuitability.Consistency 1.1.2.2 Learnability.Helpfulness.HelpCompleteness 1.2.2.1 Ease of use.Error Mgmt.Error Prevention
Effectiveness in use: Sub-task correctness	1.2.3.1 Ease of use.Data Entry Ease.Defaults 1.2.3.3 EaseofUse.DataEntryEase.ControlAppropriateness 2.1.2.2 Information quality.InfoSuitability.InfoCoverage.Completeness 2.1.2.1 Information quality.InfoSuitability.InfoCoverage. Appropriateness

The indicator mapping sets the stage for the interpreting the possible relationships between the EQ attributes and QinU attributes specified in our 2Q2U model instantiation. For example, in our mapping, we set greater than 20% improvement to *highly related* (dark gray), 5-20% to *somewhat related* (medium gray), and below 5% to *little or no relationship* (light gray). We chose this calibration as an initial benchmark to be re-calibrated and improved when more historic data is available.

As depicted, the last 2 rated attributes, *Sub-task Correctness Learnability* and *Task Successfulness Learnability*, did not improve. A possible explanation for this is that due to metric design (not shown in this paper for space reasons), with data collected over a 12 week period that the learning process did not improve as much as expected. It is possible that these beginner users possibly did not ramp up their learning during this time period and that if the case study had been longer, we may have seen different behavior and hence measurements.

By taking those attributes with a high level of improvement in QinU and mapping those high levels of improvement to the changes made in the WebApp from an EQ viewpoint gives us insight into what particular changes in EQ attributes have an effect on QinU. For instance, Table 6 shows that an EQ improvement in *Help completeness* (1.1.2.2) of 70% possibly resulted in tangible real in-use improvements for *Sub-task completeness learnability* of 50.9%. On the other hand, some EQ changes resulting in EQ improvement showed little or no QinU improvement influence. Thus, we cannot say with 100% certainty that changes made in the properties of the application

(EQ attributes) made a definite impact on a particular QinU attribute, but we can say that it had a positive influence.

Regarding the EQ and QinU attributes' relationships, we cannot quantify the exact contribution from each because we made more than one change at a time. However, if we had made only one change, and then measured QinU for JIRA v.1.1, we may have uncovered one to one relationships, although this is unlikely as it is more plausible that several EQ attributes will influence any particular QinU attribute. Table 7 summarizes these relationships developed in this case study. Each of these relationships can be further examined in future case studies.

5 Concluding Remarks

We can continue to use SIQinU with a higher level of granularity by only making one small improvement change at a time in Ph.V, and then measuring the consequence (*influences*) on QinU in Ph.VI. Our ultimate objective is to improve the QinU of WebApps-in-use. With this goal in mind, we first used 2Q2U to purposely instantiate models for both EQ and QinU as per one of our main contributions. For EQ, our model included *information quality*, a characteristic proposed in an earlier work [12] to supplement ISO 25010 to account for the particular characteristics of WebApps whose quality is dependent on information quality as well. For QinU, our model included *learnability in use*, also proposed in that work to supplement ISO 25010 to account for the time dimension of learning and for the specific task being carried out.

Using the purposefully instantiated quality models as a starting point, our second contribution is the proposed SIQinU, a consistent and repeatable strategy to improve the QinU. SIQinU uses the models generated by 2Q2U through its six phases toward evaluating a WebApp from both EQ and QinU points of view with the ultimate goal of making improvement. It also employs the C-INCAMI framework in order to guarantee consistency for usage in an iterative manner for continued improvement. Note that SIQinU relies also on a well-established process [2]. To illustrate its usage, we employed SIQinU in a case study using JIRA, a well-known defect tracking system using a task specifically designed to collect information at the sub-task level so that specific screens and their properties (EQ attributes) could be identified for potential problems leading to poor performance in QinU.

Lastly, as our final contribution, in carrying out SIQinU, we were able to map EQ characteristics and attributes to QinU attributes with the goal of ultimately achieving real improvement not only for JIRA but for WebApps and software design in general. Based on this premise, we used these relationships and the improvements made to develop a list of recommendations for improving WebApps. These relationships (currently at exploratory stage) and list of improvements can be further validated through additional case studies.

Acknowledgments. Thanks to the support by National Basic Research Program of China (No. SKLSDE-2010ZX-16), and PAE-PICT 2188 project at UNLPam, from the Science and Technology Agency, Argentina.

References

1. Basili, V., Lindvall, M., Regardie, M., Seaman, C., Heidrich, J., Munch, J., Rombach, D., Trendowicz, A.: Linking software development and business strategy through measurement. *IEEE Computer* 43(4), 57–65 (2010)
2. Becker, P., Lew, P., Olsina, L.: Strategy to Improve Quality for Software Applications: A Process View. In: To appear in ACM Proceedings of ICSE, Int'l Conference of Software and System Process (ICSSP), Honolulu, Hawaii, USA, May 21-22 (2011)
3. Bevan, N.: Extending quality in use to provide a framework for usability measurement. In: Kurosu, M. (ed.) HCD 2009. LNCS, vol. 5619, pp. 13–22. Springer, Heidelberg (2009)
4. Bevan, N., Bohomolni, I.: Incorporating user quality requirements in the software development process. In: Proceedings of 4th International Software Quality Week Europe, Brussels, pp. 1192–1204 (2000)
5. Brambilla, M., Comai, S., Frernali, P., Matera, M.: Designing Web Applications with WebML and WebRatio Web Engineering. In: Modelling and Implementing Web Applications, Ch. 9, pp. 221–261. Springer HCIS, Heidelberg (2008)
6. Burton, M., Walther, J.: The value of Web log data in use-based design and testing. *Journal of Computer-Mediated Communication* 6(3) (2001)
7. Covella, G., Olsina, L.: Assessing Quality in Use in a Consistent Way. In: ACM Proceedings, Int'l Congress on Web Engineering (ICWE 2006), SF, USA, pp. 1–8 (2006)
8. Fernandez, A., Insfran, E., Abrahão, S.: Integrating a Usability Model into Model-Driven Web Development Processes. In: Vossen, G., Long, D.D.E., Yu, J.X. (eds.) WISE 2009. LNCS, vol. 5802, pp. 497–510. Springer, Heidelberg (2009)
9. Hassenzahl, M.: User experience: towards an experiential perspective on product quality. In: Proc. 20th Int'l Conference of the Assoc. Francophone d'Interaction Homme-Machine, IHM, vol. 339, pp. 11–15 (2008)
10. ISO 13407: User centered design process for interactive systems (1998)
11. ISO/IEC 25010: Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE). System and Software Quality Models (2011)
12. Lew, P., Olsina, L., Zhang, L.: Quality, Quality in Use, Actual Usability and User Experience as Key Drivers for Web Application Evaluation. In: Benatallah, B., Casati, F., Kapel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 218–232. Springer, Heidelberg (2010)
13. Moraga, M.A., Bertoa, M.F., Morcillo, M.C., Calero, C., Vallecillo, A.: Evaluating Quality-in-Use Using Bayesian Networks. In: Proc. of QAOOSE 2008, Paphos, Cyprus (2008)
14. Nielsen, J.: Involving Stakeholders in User Testing, Jakob Nielsen's Alertbox (May 24, 2010),
<http://www.useit.com/alertbox/utest-observers.html>
(accessed in January 2011)
15. Olsina, L., Papa, F., Molina, H.: How to Measure and Evaluate Web Applications in a Consistent Way. In: Modelling and Implementing Web Applications, Ch. 13, pp. 385–420. Springer HCIS, Heidelberg (2008)
16. Olsina, L., Rossi, G., Garrido, A., Distante, D., Canfora, G.: Web Applications Refactoring and Evaluation: A Quality-Oriented Improvement Approach. *Journal of Web Engineering* 4(7), 258–280 (2008)

Reusing Web Application User-Interface Controls

Josip Maras¹, Maja Štula¹, and Jan Carlson²

¹ University of Split, Croatia

² Mälardalen Real-Time Research Center, Mälardalen University, Västerås, Sweden

{josip.maras,maja.stula}@fesb.hr, jan.carlson@mdh.se

Abstract. Highly interactive web applications that offer user experience and responsiveness of desktop applications are becoming increasingly popular. They are often composed out of visually distinctive user-interface (UI) elements that encapsulate a certain behavior – the so called UI controls. Similar controls are often used in a large number of web pages, and facilitating their reuse could offer considerable benefits. Unfortunately, because of a very short time-to-market, and a fast pace of technology development, preparing controls for reuse is usually not a primary concern. In this paper we present a semi-automatic method, and the accompanying tool, for extracting and reusing web controls. The developer selects the control and performs a series of interactions that represent the behavior he/she wishes to reuse. In the background, the execution is analyzed and all code and resources necessary for the stand-alone functioning of the control are extracted. Optionally, the user can immediately reuse the extracted control by automatically embedding it in an already existing page.

1 Introduction

In the last two decades web applications have made a tremendous leap forward: from simple static web pages developed only in HTML to complex dynamic web applications developed using server-side technologies that extensively use web services, databases and client-side technologies. Initially the term “dynamic web application” was mostly used to describe that web page content was dynamically generated on the server side. More recently (2005 and onwards) with the wide spread adoption of AJAX and faster web browsers, web applications are also increasingly dynamic on the client side (e.g. applications such as Gmail, Facebook, etc.). Web developers now routinely use sophisticated scripting languages and other active client-side technologies to provide users with rich experiences that approximate the performance of desktop applications [23].

Web application user-interface (UI) is often composed of distinctive UI elements, the so called UI controls. Similar controls are often used in different web applications and facilitating their reuse could lead to faster development. Unfortunately, the web application development domain is exposed to a very fast pace of technology development and short time-to-market. This means that preparing code for reuse is often not a primary concern. So, when developers encounter

problems that have already been solved in the past, rather than re-inventing the wheel, or spending time componentizing the already available solution (which is sometimes not preferable [8]) they perform reuse tasks [2]. Reusing source code that was not designed in a reusable fashion is known by different synonyms: copy-and-paste reuse [10], code scavenging [9] and, more recently, pragmatic-reuse [5]. Pragmatic reuse treats the system in a white-box fashion and involves extracting functionality from an existing system and reusing it within another system. The client-side web development domain is particularly pervious to white-box reuse, since code is transferred and executed in the browser. White-box reuse tasks are complex and error-prone, partly because the goal is to extract the minimum amount of code necessary for the desired functionality [5].

Reuse of client-side web UI controls is particularly difficult since there is no trivial mapping between source code and the page displayed in the browser; code is usually scattered between several files and code responsible for the desired functionality is often intermixed with code irrelevant for the reuse task. In order to reuse the chosen control, the developer has to locate the code and the resources defining the UI control. Next, the developer has to download the selected files, remove the unnecessary code and resources, and adjust for the now changed location. This is a time-consuming process.

The structure of a web page is defined by HTML code, the presentation by CSS (Cascading Style Sheets) code, and the behavior by JavaScript code. In addition, a web page usually contains various resources such as images or fonts. The interplay of these four basic elements produces the end result displayed in the user's web browser. Visually and behaviorally a web page can be viewed as a collection of UI controls, where each control is defined by a combination of HTML, CSS, JavaScript and resources (images, videos, fonts, etc.) that are intermixed with code and resources defining other parts of the web page.

In this paper we present a novel approach to semi-automatic extraction of reusable client-side controls. The developer selects the desired UI control on the web page and interacts with it, demonstrating the behavior that he/she wishes to reuse. In the background, the tool that we have developed – Firecrow [12] tracks all executed code, applied CSS styles and used resources, in order to locate the code and resources that are vital for the stand-alone functioning of the chosen UI control. In the end, all essential code and resources are extracted, all necessary adjustments are made and the control is packed as a reuse-friendly web page. Optionally, the developer can choose to embed the extracted UI control directly into a specific place of an already existing web page.

2 Extracting and Reusing UI Controls

In order to reuse a web UI control, we have to extract all that is necessary for the control to be visually and functionally autonomous. This means extracting all HTML, CSS, JavaScript and resources that are used in the visual presentation and the desired behavior of the control.

The process can be separated into three phases: 1) Interaction recording, 2) Resource extraction, and 3) UI control reuse (Figure 1).

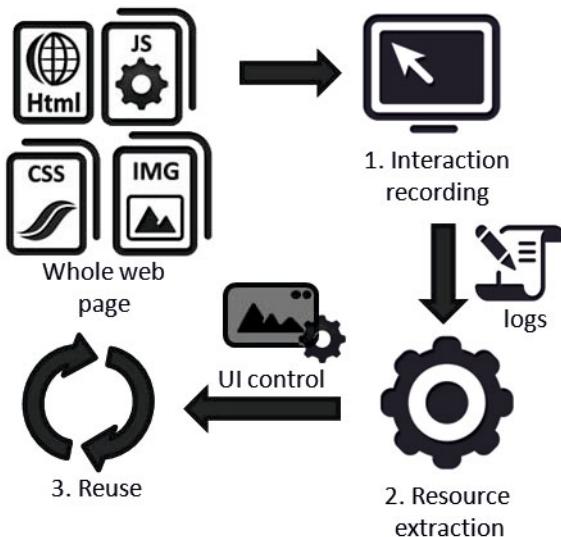


Fig. 1. Extracting and reusing UI controls in Web applications

The first step of the *Interaction recording* phase is to select the HTML node that defines the chosen UI control. Next, the user performs a series of interactions that represent the behavior of the control. The purpose of this phase is to gather a log of all resources required for replicating visual and behavioral aspects of the control.

The life-time of the web application client-side can be divided into two steps: *i)* page initialization – where the browser parses the web page code and builds the DOM (Document Object Model) [19] of the page, and *ii)* event-handling. All user interactions are handled in the event-handling phase by modifying the DOM built in the initialization phase. This means that in the beginning of the recording phase, before the user has started interacting with the UI control, a “snapshot” of the initial state of the control has to be made. This is done by logging all executed initialization code, all CSS styles and all resources used to initially define the control. Later, all code executed during the recorded interaction is also logged, together with all dynamically applied CSS styles and images.

When the user chooses to end the recording, the process enters the *Resource extraction* phase, where code models for all code files (HTML, CSS, and JavaScript) are build. Based on those models and logs gathered during the recording phase, the code necessary for replicating the visuals and the demonstrated behavior is extracted.

After the extraction phase is finished, the user can choose to enter the *Reuse* phase and automatically embed the extracted control in an existing web page, either by replacing, or by embedding it inside an already existing node. In this way a full cycle is completed: from seeing the potential for reuse, through extracting the desired control, all the way to actually reusing it and gaining new functionalities in the target web page. Each step of the process is described in more detail in the following sections.

The approach will be illustrated with an example of extracting and reusing a UI control from a web page. Figure 2 shows a web page developed in a previous project, and the control (marked with a dashed frame) that was selected for reuse. The control displays different images (marked with 1 in Figure 2) and captions (mark 2). Currently displayed items can be changed by clicking on bullets (mark 3), and the control replaces items with a fade-out, fade-in effect.

3 Interaction Recording

The purpose of the interaction recording phase is to locate all code and resources necessary for stand-alone functioning of the target UI control. In order to do that, the user has to first select the chosen control. However, the control does not exist as a separate entity in the web page. So, in our approach the control is selected through the corresponding HTML node defining the UI control. This is done with Firebug's DOM (Document Object Model) explorer in which the user can go through the DOM of the page.

When the user initiates the recording phase, the page is reloaded, subscriptions to the DOM mutation events [20] are registered, and the initial state of the UI control is logged. The initial state is composed of code executed while initializing the control, and styles and resources that initially define the control. Generally, all executed code is logged by communicating with the JavaScript debugger service, which provides hooks (or events) that activate on each execution and give information about the currently executed source code lines. In order to obtain the styles and resources that initially define the UI control, the DOM of the control is traversed and all CSS styles and resources applied and used in the control are logged. With this, a log of resources that initially define the control is obtained.

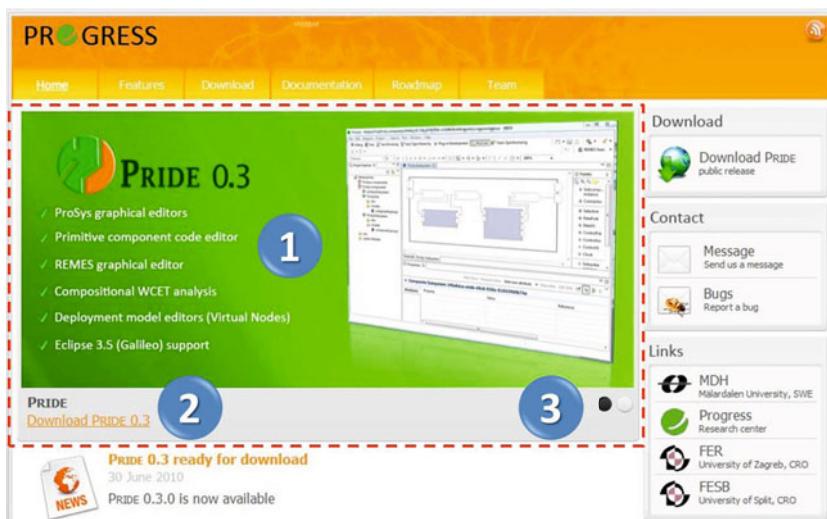


Fig. 2. The web page of the UI control chosen for extraction

After the UI control is fully loaded, the modifications of the control are caused by user interactions and/or timing events. The executed code is again logged by communicating with the JavaScript debugger service, while any dynamic changes in styles and resources are logged by handling DOM mutation events. Using this approach we are able to locate all code and resources that define the control: *i*) HTML code, because the user directly selects the HTML node defining the control; *ii*) JavaScript code, because by communicating with the JavaScript debugger service we are able to log all executed lines; *iii*) CSS code; and *iv*) resources, because styles and resources applied to the control during the whole course of the execution are logged.

4 Extraction

Once the recording of interactions is complete, the process goes into the extraction phase. As input, the extraction process receives all data gathered during the recording phase: the HTML code of the whole page; the xPath [21] expression uniquely defining the node designated for extraction; a collection of used CSS styles and resources; and for each JavaScript code file a list of executed lines. Based on this data, JavaScript files, CSS files, and resources are separately analyzed and cleansed of unnecessary elements. Since HTML documents can have JavaScript and CSS code embedded directly in them, these parts of the file are handled in the same way as the rest of the JavaScript and CSS code.

4.1 Extracting JavaScript Code

The goal of JavaScript code extraction is to produce a minimal code that is syntactically correct, and semantically consistent with the recorded execution. A naive implementation, where one would simply split the file into lines, and then filter out the non-executed ones can only function in rare cases of specially formatted code, but since real-world code can be arbitrarily formatted this is not an option. Consider the example given in Listing 1.1.

```
/*1*/var a = getNum();
/*2*/if(a%2==0) {doEven();} else
/*3*/{
/*4*/    doOdd();
/*5*/
/*6*/doOtherStuff();
```

Listing 1.1. Why naive line removal does not work

In this example, if the *getNum* function returns an even number, lines 1, 2 and 6 will be executed. If we do simple line removal, we would end up with code presented in Listing 1.2 which is not semantically equivalent to the original program. The *doOtherStuff* function would only be called if *a* is odd. And in the original code (Listing 1.1) it is called regardless if *a* is even or odd.

```
/*1*/var a = getNum();
/*2*/if(a%2==0) {doEven();} else
/*3*/doOtherStuff();
```

Listing 1.2. Naive line removal result

In order to tackle this problem, we build a model of the JavaScript source code. This model is produced by a JavaScript parser that we have developed. The parser is developed with ANTLR [7] according to the specification given in [6]. Listing 1.3 gives a code example and Listing 1.4 gives the model generated from the code example.

```
/*1*/function double(x)
/*2*/{
/*3*/    return 2*x;
/*4*/}
```

Listing 1.3. JavaScript code example

As can be seen in Listing 1.4, the model represents a simplified abstract syntax tree of the given source code. Although it is a lot more verbose than the source code from which it is derived from, the model provides all information about the position and type of used constructs.

```
{"srcElems": [
  "type": "funcDecl", "strtLn": 1, "strtCh": 0,
  "name": "double", "params": ["x"],
  "body": {
    "type": "funcBody",
    "strtLn": 2, "strtCh": 0, "srcElems": [
      "type": "rtrnStatement", "strtLn": 3, "strtCh": 1,
      "expr": {
        "type": "mulExpr", "strtLn": 3,
        "strtCh": 8, "exprs": [
          {"type": "numLit", "strtLn": 3,
           "strtCh": 8, "value": 2,
           "endLn": 3, "endCh": 8},
          {"type": "mulExprItem", "strtLn": 3,
           "strtCh": 9, "operator": "*",
           "item": {
             "type": "ident", "strtLn": 3, "strtCh": 10,
             "id": "x", "endLn": 3, "endCh": 10},
           "endLn": 3, "endCh": 10
         }],
        "endLn": 3, "endCh": 10
      },
      "endLn": 3, "endChar": 11
    ],
    "endLn": "4", "endCh": "0"
  },
  "endLn": 4, "endCh": 0
}]}
```

Listing 1.4. JavaScript model example

By traversing a source code model (e.g. Listing 1.4) we can remove all code constructs contained in the not-executed lines, while keeping the semantical correctness.

4.2 Extracting CSS Code

Part of the data gathered during the recording phase are styles that get applied to the chosen HTML node. Similarly to the process of building the JavaScript code model, we have also developed a CSS parser that builds a model of the code. Listing 1.5 gives a code example, while Listing 1.6 presents the model of the code given in the example.

```
@import url('/css/style.css');
body { font-family: tahome; background: white; }
```

Listing 1.5. CSS code example

Based on the used CSS styles that were gathered during the recording phase, the code model of the whole web application CSS code is traversed and only code comprised of used styles is generated.

```
{ "imports": [ { "url": "/css/style.css" } ],
  "body": { "items": [
    { "type": "ruleSet",
      "selectors": ["#mainContainer"],
      "declarations": [
        { "prop": "font-family", "val": "tahoma" },
        { "prop": "background", "val": "white" }
      ] } ] }}
```

Listing 1.6. CSS model example

The CSS code model is especially useful in the reuse phase, where the CSS code of the control is merged with the CSS code of the host web page. There we use both the CSS model of the control and the CSS model of the host page to detect naming conflicts, and in the case of conflicting styles offer the possibility of merging.

4.3 Extracting HTML Code

In order to extract the HTML code of the chosen UI control, we have to be able to locate the code responsible for defining the control. In this case we build a standard model of the given web page – DOM [19] – with an open source HTML parser [14].

The visual layout of a certain HTML node is not only influenced by the HTML code of that node, but also by the type and presentation of its ancestors. For this reason, when traversing the DOM tree, the ancestors of the UI control

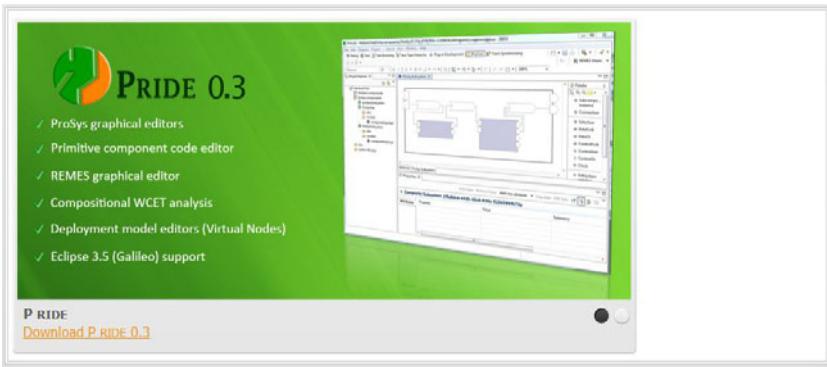


Fig. 3. Extracted UI control from the web page shown in Figure 2

are kept, but all siblings (both from the chosen HTML node, and from each of its ancestors) are removed. The resulting HTML DOM tree can contain image nodes and references to styles and scripts. Since the location of the HTML node will change (the code will be transferred from a web server to the user chosen location), the in-code references to those files also have to be changed. This is a time-consuming task, so it is handled automatically.

4.4 Extracting Resources

During the recording phase, all resources that were at some point used by the UI control are tracked. In the final step of the extraction phase all resources are automatically downloaded to the target location.

The result of the extraction process is an HTML document that contains the HTML code defining the selected node, and includes all necessary style sheets, scripts, and resources that were identified as necessary in the extraction phase. Figure 3 shows the results of extracting the UI control from the web page shown in Figure 2.

5 Reuse

Once the UI control has been extracted it can be embedded into an existing “host” web page. The user selects the host web page, a referent node in the host page, and the insertion type. The control can be inserted so that it replaces the referent node; or it can be inserted into, before, or after the referent node. In order to enable reuse, resources defining the extracted UI control and resources defining the host web page have to be merged. Naturally, this can lead to conflicts such as CSS style overriding, duplicate JavaScript libraries, name clashes, etc., that have to be tackled.

5.1 Detecting Conflicts

Currently, the best reuse results are achieved if the extracted UI control is reused at an early stage of the new web page development – in a state where the host

web page is not complex and does not include large CSS or JavaScript code bases. Even in that case, conflicts when merging HTML, CSS, JavaScript can occur.

Detecting HTML conflicts – When merging the HTML code of the control with the HTML code of the host web page the following conflicts can occur: inclusion of duplicate JavaScript libraries, occurrence of HTML nodes with the same id, and clashing HTML node classes. Before the merging is done, the control DOM and the host page DOM are analyzed, and if any conflicts are detected the user is notified.

Detecting CSS conflicts – In CSS, conflicts can arise from clashes based on HTML node IDs, node classes and node types. The first two cases (IDs and classes) are handled by detecting HTML conflicts, but the third case has to be handled separately. If there are CSS rules clashing because of node types, then we notify the user and offer the possibility to either accept one of the rules, or to merge them into one CSS rule.

Detecting JavaScript conflicts – Web applications often use JavaScript libraries (e.g. jQuery, Prototype, Mootools, etc.), and a situation might happen in which the host page uses a full library, and the control uses a subset of the code from the same library, or vice versa. In that case we handle the conflict by including the full library. Since JavaScript is a dynamic language, more advanced analysis (that is beyond the scope of this paper) is needed in order to detect conflicts on variable or function level.

5.2 Example

We will demonstrate the process by reusing the UI control, described in Section II, in a test web page shown in Figure 4. When extracting the control we provide the path to the test host web page, and the xPath expression of the placeholder node which we want to replace. Then, when the control is extracted all control resources are merged with the already existing resources of the host page.

The result is shown in Figure 5. A video showing the whole process of reuse, can be found at the Firecrow web page¹.

In order to complete the reuse and adapt the extracted UI control to the new context, the developer has to manually replace some of the extracted resources. For example, in this case we have replaced background images, changed text captions, and added another option (the result is shown in Figure 6).

6 Tool

The whole process is currently supported by the Firecrow tool [12], which is an extension for the Firebug² web debugger. Currently, the tool can be used from the Firefox web browser, but it can be ported to any other web browser that

¹ <http://www.fesb.hr/~jomaras/Firecrow>

² <http://getfirebug.com>

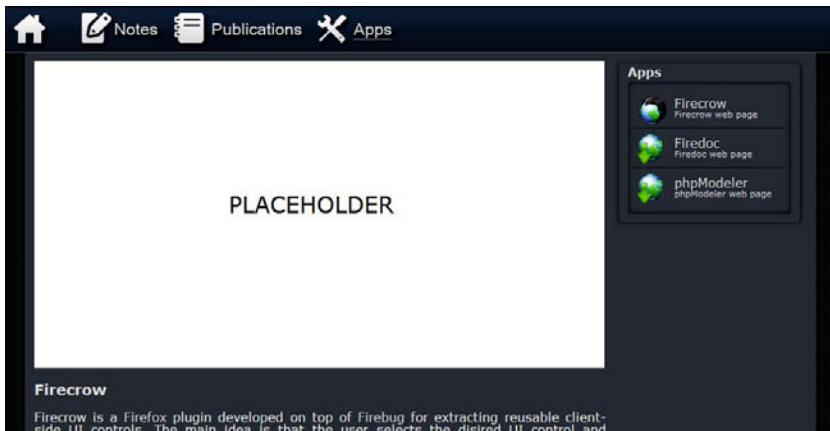


Fig. 4. The host web page with a placeholder for the extracted control

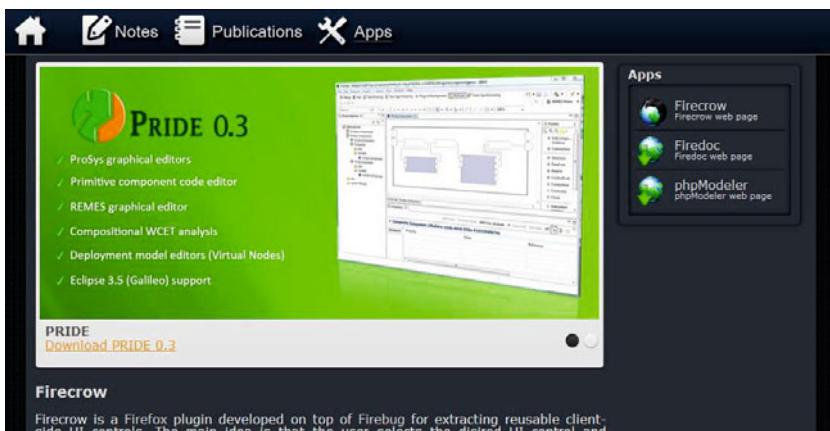


Fig. 5. The result of reusing UI control in the host web page

provides communication with a JavaScript debugger, and a DOM explorer (e.g. IE, Chrome, Opera). Only the interaction recording phase is browser dependent; building source models, extracting code, downloading resources, merging code and resources are functionalities that are all encapsulated in a Java library that can be called from any browser on any operating system. The whole source of the program can be downloaded from <http://www.fesb.hr/~jomaras/?id=Firecrow>.

The Firecrow UI is shown in Figure 7. Mark 1 shows the web page chosen for extraction, mark 2 Firebug's HTML panel used for selecting controls, and mark 3 Firecrow's extraction and reuse wizard.



Fig. 6. The result of adapting the extracted UI control

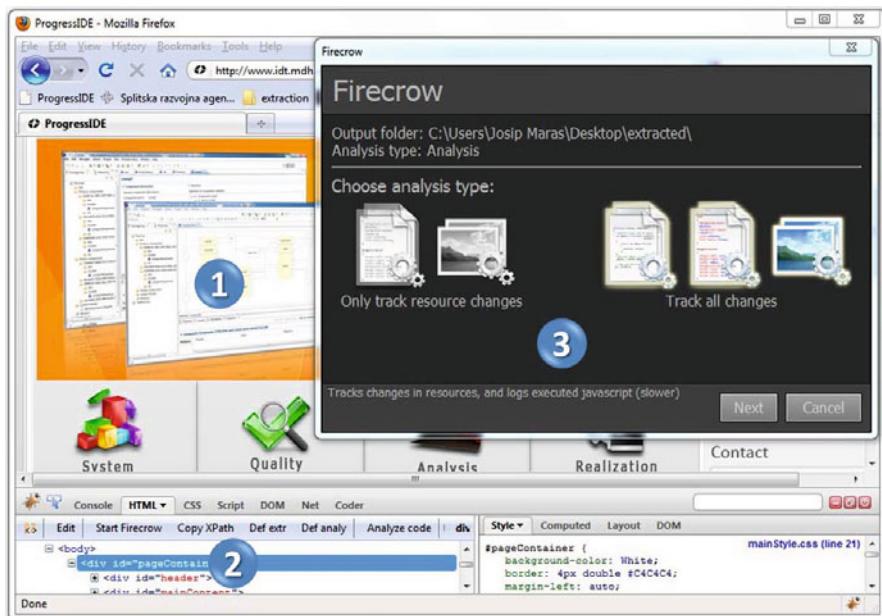


Fig. 7. Firecrow user interface

7 Evaluation and Lessons Learned

We have evaluated our approach by extracting UI controls from thirty-five web pages: twenty have been selected from the top 200 most visited web pages in the world [1] (including Google, Facebook, Twitter, and Apple), ten have been selected because they have visually interesting controls, and five from projects in

which we have been involved earlier. The full list of tested web pages is available on the Firecrow web page and is updated regularly as more web pages are tested. Since the notion of what would be considered a user control can vary, each web page is accompanied with a screen-shot marking the extracted user controls.

In this evaluation we have learned more about the advantages and shortcomings of the approach and the accompanying tool Firecrow. From thirty-five web pages we were able to successfully extract 133 user controls. However, the extraction of eleven user controls failed. Mostly, the problems were with HTML parsing, heavily modified DOM, and JavaScript trying to access elements that were deleted in the extraction process. Problems with parsing HTML arise from the fact that browsers fix invalid HTML code, and since there is no standard way of handling HTML errors, each browser handles this problem in a specific way. We use an open source HTML parser, and the DOM produced by this parser does not always match the DOM built by a browser. Since we are identifying nodes by xPath expressions which define the nodes' position in the DOM, in some rare cases, there can be a mismatch between the position of the node in the browser DOM and in the DOM built by the parser. Because of this, the extraction process can not locate the node, and the extraction fails. Also, a node can not be located if the chosen node was dynamically created and does not exist in the original HTML defining the web page.

In the current approach, we extract all code executed while loading the page and while executing control-specific behavior. This means that we will usually end up with more code than is actually needed for the user-control behavior (e.g. initialization code for other controls can be executed). Some of that code can try to access web page elements that are deleted in the extraction process, which in turn can cause JavaScript errors in the extracted web page. Detecting these errors with the Firebug web debugger, is however fairly straightforward.

8 Related Work

There exist a number of approaches, environments and tools designed to support reuse. In the web application domain these include HunterGatherer [15], Internet Scrapbook [16], HTMLviewPad [17], and ReWeb [18]; while in the more general domain of reusing Java code there is G&P (Gilligan and Procrustes) [4].

HunterGatherer [15] and Internet Scrapbook [16] allow users to collect components from within Web pages, and to collect components from different Web pages into a newly created page. But since these approaches were developed in 1990's and early 2000, when web page development was not so dynamic on the client side, with the term "component" they refer to information components – most usually text paragraphs. These approaches are mostly used to create scrapbooks of data gathered from different web pages, and not to reuse certain functionality and visual elements of web pages.

Tanaka et al. [17] describe an interesting approach to clipping and reusing fragments of Web pages in order to compose new applications. They only target HTML elements, specifically HTML forms (no attention to CSS or JavaScript

is given in their examples), and how to reroute data entered in the form to original servers that process the request. The applications created in this way are not deployable as standard web pages, but are executed within their tool – HTMLviewPad. This fact, along with not explicitly targeting the whole technology chain (HTML, CSS, and JavaScript) is the biggest difference between our approaches.

Our work is also related to program slicing [22], where by starting from a subset of a program's behavior, the program is reduced to a minimal form which still produces that behavior. In a sense our approach can be viewed as web page slicing with the goal of reducing the whole page (along with its code and resources) to a form in which only the visuals and the behavior of the selected user control are maintained. In the web engineering domain Tonella and Ricca [18] define web application slicing as a process which results in a portion of the web application which exhibits the same behavior as the initial web application in terms of information of interest displayed to the user. In the same work they present a technique for web application slicing in the presence of dynamic code generation where they show how to build a system dependency graph for web applications. This work is mostly dealing with reusing HTML and server-side code.

In the more general domain of Java applications, G&P [4] is a reuse environment composed of two tools: Gilligan and Procrustes, that facilitates pragmatic reuse tasks. Gilligan allows the developer to investigate dependencies from a desired functionality and to construct a plan about their reuse, while Procrustes automatically extracts the relevant code from the originating system, transforms it to minimize the compilation errors and inserts it into the developer's system. This work was further expanded [3] with the possibility to automatically recommend elements to be reused based on their structural relevance and cost-of-reuse.

There are also two tools that facilitate the understanding of dynamic web page behavior: Script InSight [11] and FireCrystal [13]. Script InSight helps to relate the elements in the browser with the lower-level JavaScript syntax. It uses the information gathered during the script's execution to build a dynamic, context-sensitive, control-flow model that provides feedback to the developers as a summary of tracing information. FireCrystal [13] is a standalone Firefox plug-in that facilitates the understanding of interactive behaviors in dynamic web pages. FireCrystal performs this functionality by recording interactions and logging information about DOM changes, user input events, and JavaScript executions. After the recording phase is over, the user can use an execution time-line to see the code that is of interest for the particular behavior. Compared to our approach they make no attempts to extract the analyzed code.

9 Conclusion and Future Work

In this paper we have presented a novel approach and the accompanying tool for extracting and reusing client-side user interface controls in web applications. The process starts with the developer selecting the user control and demonstrating the behavior that he/she wishes to reuse. In the background, the executed code

and used resources are analyzed. We have shown how, based on that analysis, a subset of the whole application code and resources necessary for the independent functioning of the control can be determined. We have evaluated the approach on thirty-five web applications and found that in a majority of cases the process is able to extract stand-alone UI controls.

During the evaluation we have noticed that some pages make extensive modifications of the original web page DOM, up to the point that the node defining the UI control does not exist in the original HTML code. Also, even though some statements get executed while interacting with the control, they are not necessarily required for the functioning of the web control. So, for future work, we plan to develop a method for tracking DOM changes, which should enable us to locate nodes in the original HTML code required for the creation of the node defining the UI control. We also plan to extend the analysis of executed code, so that only code statements that influence the behavior of the control are extracted.

The web page displayed in the browser is usually the result of server-side program execution, so we plan to extend the approach with server-side code analysis in order to facilitate code reuse on the server-side.

Acknowledgment

This work was partially supported by the Swedish Foundation for Strategic Research via the strategic research center PROGRESS, and the Unity Through Knowledge Fund supported by the Croatian Government and the World Bank via the DICES project.

References

1. Alexa: Alexa top sites (October 2010), <http://www.alexa.com/topsites/>
2. Brandt, J., Guo, P.J., Lewenstein, J., Klemmer, S.R.: Opportunistic programming: How rapid ideation and prototyping occur in practice. In: WEUSE 2008: Workshop on End-User Software Engineering, pp. 1–5. ACM, New York (2008)
3. Holmes, R., Ratchford, T., Robillard, M.P., Walker, R.J.: Automatically Recommending Triage Decisions for Pragmatic Reuse Tasks. In: ASE 2009: Proceedings of the 2009 24th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, Los Alamitos (2009)
4. Holmes, R., Walker, R.J.: Semi-Automating Pragmatic Reuse Tasks. In: ASE 2008: Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, pp. 481–482. IEEE Computer Society, Los Alamitos (2008)
5. Holmes, R.: Pragmatic Software Reuse. PhD thesis, University of Calgary, Canada (2008)
6. ECMA: international. ECMAScript language specification, <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf>
7. Bovet, J.: Antlr web site (February 2011), <http://www.antlr.org/>

8. Kapser, C., Godfrey, M.W.: "Cloning Considered Harmful" Considered Harmful. In: WCRE 2006: Proceedings of the 13th Working Conference on Reverse Engineering, pp. 19–28. IEEE Computer Society, Los Alamitos (2006)
9. Krueger, C.W.: Software reuse. ACM Comput. Surv. 24(2), 131–183 (1992)
10. Lange, B.M., Moher, T.G.: Some strategies of reuse in an object-oriented programming environment. In: SIGCHI Bull., vol. 20(SI), pp. 69–73
11. Li, P., Wohlstädter, E.: Script InSight: Using Models to Explore JavaScript Code from the Browser View. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 260–274. Springer, Heidelberg (2009)
12. Maras, J., Štula, M., Carlson, J.: Extracting Client-Side Web User Interface Controls. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 502–505. Springer, Heidelberg (2010)
13. Oney, S., Myers, B.: FireCrystal: Understanding interactive behaviors in dynamic web pages. In: VLHCC 2009: Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 105–108. IEEE Computer Society, Los Alamitos (2009)
14. Open source Tagsoup. Tagsoup, (September 2010) <http://home.ccil.org/~cowan/XML/tagsoup/>
15. Schraefel, M.C., Zhu, Y., Modjeska, D., Wigdor, D., Zhao, S.: Hunter Gatherer: Interaction Support for the Creation and Management of Within-Web-Page Collections. In: 11th International Conference on World Wide Web, pp. 172–181 (2002)
16. Sugiura, A., Koseki, Y.: Internet scrapbook: creating personalized world wide web pages. In: CHI 1997: Extended Abstracts on Human Factors in Computing Systems, pp. 343–344. ACM, New York (1997)
17. Tanaka, Y., Ito, K., Fujima, J.: Meme Media for Clipping and Combining Web Resources. World Wide Web 9, 117–142 (2006)
18. Tonella, P., Ricca, F.: Web Application Slicing in Presence of Dynamic Code Generation. Automated Software Engg. 12(2), 259–288 (2005)
19. World Wide Web Consortium (W3C). Document Object Model (DOM) (September 2010), <http://www.w3.org/DOM/>
20. World Wide Web Consortium (W3C). Document Object Model Events (September 2010), <http://www.w3.org/TR/DOM-Level-2-Events/events.html>
21. World Wide Web Consortium (W3C). XML path language (xpath) (September 2010), <http://www.w3.org/TR>xpath/>
22. Weiser, M.: Program slicing. In: ICSE 1981: 5th International Conference on Software Engineering, pp. 439–449. IEEE Press, Los Alamitos (1981)
23. Wright, A.: Ready for a Web OS? Commun. ACM 52(12), 16–17 (2009)

Tools and Architectural Support for Crowdsourced Adaptation of Web Interfaces

Michael Nebeling and Moira C. Norrie

Institute of Information Systems, ETH Zurich
CH-8092 Zurich, Switzerland
{nebeling,norrie}@inf.ethz.ch

Abstract. There is a vast body of research dealing with the development of context-aware web applications that can adapt to different screen and user contexts. However, the range and growing diversity of devices used for web access makes it increasingly difficult for developers to provide a design and layout that adapts well to every client. To address this, we propose a crowdsourcing approach that allows developers to define a default web interface suitable for many devices and enables the crowd, i.e. other developers or even non-technical end-users, to adapt it to particular use contexts. We present an architecture for creating and sharing adaptations as well as suggesting and applying these in matching contexts. In addition, we discuss the underlying crowdsourcing principles and present a set of visual tools that facilitate the adaptation process.

1 Introduction

We are currently in a period where there is a proliferation of new devices with very different characteristics in terms of, not only screen size and resolution, but also input and output modalities. It is becoming increasingly difficult for application developers to cater for these in a responsive manner. For example, the emergence of new tablet computers such as Apple's iPad and tabletop systems such as Microsoft's Surface requires applications to adapt to much larger forms of touch screens than offered by mobile phones, but many applications on the iPad still use an interface developed for the iPhone and therefore do not take advantage of the greater screen space. When it comes to large interactive surfaces, most application developers, including web site providers, have not even considered these despite the fact that they are now becoming commonplace in offices, meeting rooms, public spaces and homes. In addition, with the means for direct touch input and support for gesture-based modalities on this new generation of multi-touch devices, the trend is also towards user interfaces that promote more natural interactions and are therefore required to scale according to a user's motor skills and potential impairments. We believe that the only way that application developers can cater for such a diverse and rapidly evolving range of use contexts is to adopt a crowdsourcing model where they provide a kernel application and other developers, or even non-technical end-users, create

and share adaptations using visual tools in order to support particular devices and preferences.

In this paper, we present an architecture that enables user-driven web site evolution in terms of both the context-awareness and adaptive behaviour of the system through user participation. The approach is based on a context-aware toolkit that integrates with existing web pages so that users can create adaptations directly in the browser. These adaptations are then deployed on a server and automatically downloaded and applied the next time that the web site is accessed from a matching client. The main technical contributions are (1) an extension of the common web application architecture that facilitates crowdsourced web site adaptations, (2) a visual toolset for performing the adaptations that works with existing web sites and does not impose certain conventions on the hypertext specification and (3) a lightweight implementation that builds on only CSS and CSS media queries to support the adaptation process directly in the browser.

We begin in Sect. 2 with a discussion of the background and then provide an overview of the approach in Sect. 3. The proposed adaptation operations are presented in Sect. 4, followed by the crowdsourcing architecture in Sect. 5 and implementation details in Sect. 6. Finally, we discuss the crowdsourcing approach in Sect. 7 and give concluding remarks in Sect. 8.

2 Background

Crowdsourcing has become a popular technique for activating user communities and allowing them to contribute their experience and knowledge [1]. This is generally done by providing a simple interface to access and extend content or functionality and often by turning users into developers [2]. Two of the best known examples of crowdsourced web platforms are Facebook¹ and WordPress². Many parts of these platforms have been developed by volunteers and shared with users in the form of small applications, plugins or themes that can simply be installed and hooked into the running system to extend its functionality. Our aim is to develop mechanisms whereby crowdsourcing can be integrated into existing web applications in order to collectively solve problems of design deficiencies with respect to different screen and user contexts through user participation.

There are two main approaches to supporting adaptivity in applications—*adaptive interfaces* and *adaptable interfaces*. An adaptive interface dynamically adapts the application to support the current viewing context. In contrast, an adaptable interface provides means for customisation, primarily relying on the user to employ the mechanisms to adapt the interface. The fundamental difference is who is in control of the adaptation process: adaptive interfaces are system-controlled relying on context information provided to the application, whereas adaptable interfaces are user-controlled and therefore require user intervention [3]. To cater for the proliferation of different devices used for web

¹ <http://www.facebook.com>

² <http://www.wordpress.org>

browsing, we propose a novel crowdsourcing approach that makes use of both techniques in that system developers can provide an adaptive interface that initially caters for the most common use contexts, but adaptive features can then evolve at runtime with the help of users who can refine the adaptations to better match their particular use context.

The adaptation of web interfaces to devices is often dealt with in web engineering as one particular aspect of context-awareness and efforts have been made to uniformly address the requirements of personalisation, internationalisation and multi-channel access. For example, for web design methodologies such as WebML and Hera, extensions have been proposed for modelling the behaviour of context-aware sites, e.g. [4,5]. In general, when a context-aware page is requested, the adaptation operations are executed in order to adapt the content, navigation and presentation according to the client context. For the specification of adaptation operations, both rule-based approaches [6] following the Event-Condition-Action paradigm and aspect-oriented techniques [7] that promote the use of aspects to achieve a systematic separation of general system functionality and context-aware adaptation are popular.

A second stream of research on model-based user interfaces has spawned a number of frameworks and tools which separate out several levels of user interface abstraction to be able to adapt to different user, platform and environment contexts, e.g. CAMELEON [8]. The suggested development processes typically unfold along a four-step, top-down approach, starting with domain concepts and task modelling, followed by subsequent transformation steps from abstract to concrete and the final user interfaces. The authoring of adaptive and multimodal user interfaces has also been the subject of extensive research, e.g. [9]; however, in reality, many steps in the promoted top-down engineering process are often skipped by web developers who tend to directly produce the final interfaces through a series of rapid prototypes using rich WYSIWYG tools such as Photoshop or Flash.

As an alternative, we have proposed an approach based on a domain-specific language and a runtime platform using versioning principles and a multi-variant component model [10] to support the development of context-aware web sites. The essence of this language-based approach is to support multiple application-specific aspects of context-awareness by specifying adaptations along a combination of clearly defined context dimensions and states, and to then use a context algebra that allows for powerful matching expressions to support variations of the final interface at the different levels of content, structure and layout depending on the context.

While we acknowledge that all these approaches to context-aware adaptation are very systematic and therefore good at supporting developers in the definition and deployment of adaptations, they primarily target developer-specified adaptation and come at the price of increased complexity and therefore costs. For the proposed crowdsourcing approach, we aim to abstract from underlying models and languages by providing visual tools for the adaptation of the final interface directly in the browser, which may not only increase the productivity

of developers, but is potentially also more attractive to a much larger group of non-technical users.

3 Approach

The main goal of our work is to augment developer-specified web interfaces with user-contributed adaptations to cater for a much wider range of use contexts to which applications can adapt. This crowdsourcing model involves a double role of users—one that sees the user as the usual consumer that merely benefits from shared adaptations and the other that turns them into active contributors and allows them to define and deploy adaptations that better match particular client contexts. If users choose to contribute, they can adapt the web interface using our tools that providers can integrate and bundle with a web site or offer separately in the form of a browser plugin. In the first case, created adaptations are managed by the web site and directly shared with consumers, while in the second case they are deployed as part of a separate service where users can take the initiative and create new adaptations and share them even across sites. For the latter scenario, we would like to follow the popular examples of programmableweb.com and userscripts.org, where already large communities of active users maintain shared collections of web mashups and augmentations.

From a technical point of view, our approach builds on the general web architecture where the browser renders the web interface from HTML holding the content and CSS defining the format and style as returned by the server in response to client requests. The principal idea is that all adaptations are essentially represented as modifications of the CSS that can additionally be downloaded for the original web site. This makes for a lightweight adaptation technique as no additional versions of the HTML document must be maintained and also web site generation is not required contrary to many other approaches. Another benefit of using only features of CSS to represent the adaptations is that the actual adaptation process can run completely on the client-side simply by linking the adapted CSS to the web document and hence does not depend on more expensive server-side computation. Also important is the fact that CSS definitions can be cascaded with the consequence that the layout of elements can be adjusted in multiple steps by building on from previous definitions. For users this means that adaptations do not always have to be defined from scratch, but can easily be based on other users' contributions that mostly need to be refined, and therefore tends to require less effort of the end-user.

Web sites that could benefit the most from our crowdsourcing idea are those that typically attract large numbers of users and support them in carrying out a real task. Failure to adapt to the particular screen and user contexts is then perceived as especially disturbing and counter-productive to achieving the task in an efficient way. To give an example, news web sites represent one particular type of application that is typically accessed from a range of different devices. The main task is to provide up-to-date news content and support users in accessing this often text-heavy information: special attention must therefore be paid to all

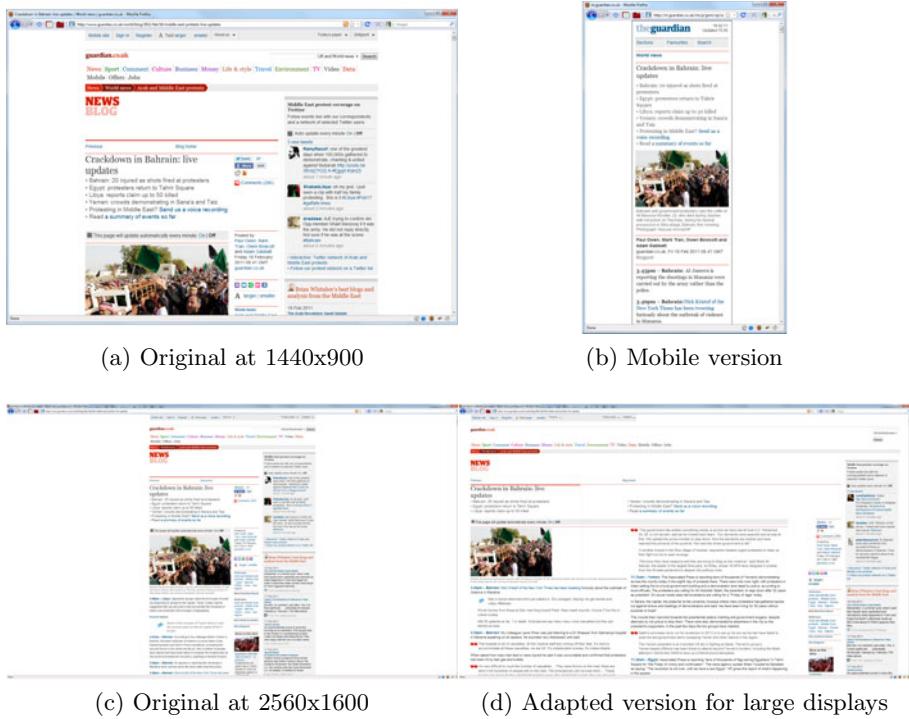


Fig. 1. Examples showing the Guardian’s news web site in (a) its original design and (b) the mobile version; (c) demonstrates the relatively poor viewing situation on a large screen and (d) shows adaptations created using our tools to make more effective use of the screen with multi-column layout

factors concerning on-screen readability. Although there is an increasing trend to provide special, mobile versions in order to manage with limited resources and computing power, there is often still a mismatch due to the great diversity in terms of screen resolutions and input/output modalities of hand-held devices. Moreover, looking at the other end of the spectrum, the user experience on large, wide-format screens often does not improve despite the fact that a much larger screen space is available. The main reason for this is a static, fixed-width design of many web sites which let content flow primarily in the vertical direction and therefore tend to leave large parts of the screen unused and, as a result of this, impose unnecessary scrolling. In a recent study [11], we have investigated news site content layout in large-screen contexts and shown that the majority of sites do not adapt well at resolutions above 1024x768 pixels despite the obvious trend towards resolutions much higher than that.

Figure 1 shows the Guardian’s online newspaper as one example of a very common news site design viewed in different settings. The site comprises the typical web page elements, such as the header containing the main navigation bar at the top, followed by the main content in three columns and the footer at

the very bottom of the page. The leftmost column is the largest and contains the news article content which typically consists of a heading, a picture and the article text itself. The other columns contain a combination of advertisements, related pages and various other services. The site is best viewed at resolutions around 1024x768 as illustrated in Fig. 1a. The mobile version offered by the web site is shown in Fig. 1b at a resolution of 640x960, as an example of how it would view on an iPhone and similar devices. Already this setting could benefit from smaller adaptations in order to use the entire width. To demonstrate that the fixed design is even more problematic in a widescreen setting, Fig. 1c shows the web site in its original design viewed on a 30" screen at a resolution of 2560x1600. To make more effective use of the much larger screen real estate similar to Fig. 1d, our tools would allow a user to define new adaptations suited to the widescreen format by resizing the main content container to fill most of the space available and realigning the inside elements. Additionally, multi-column layout could be used for both the main content area and the sidebar to automatically divide the now relatively large content areas into smaller portions. As a result, paragraphs are of appropriate width rather than producing excessively long lines, also showing more content and related navigation options on the first screen and therefore reducing the amount of scrolling necessary.

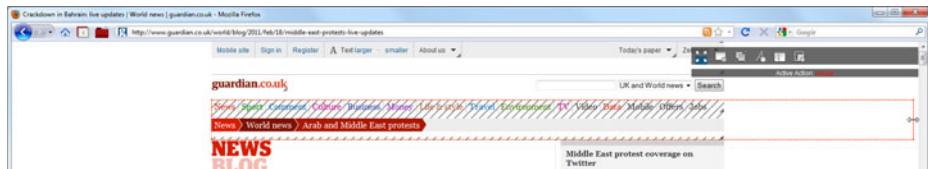
4 Adaptation Operations

To support users in carrying out the adaptation process, we build on a simple set of tools that are loaded into the browser and work directly on the visual representation of the web document. We have defined the set of adaptation operations summarised in Table 1 by looking at which adaptations are required to make more effective use of particularly small and large screens for many web sites. The examples that we considered were not constrained to news sites only, but ranged from blogs, wikis, forums to other forms of applications that are typically used by active user communities for both the consumption of content and content contribution. The defined operations, especially when used in combination, cater for a wide range of adaptations, but also reflect what is technically feasible without imposing a particular model of web site design.

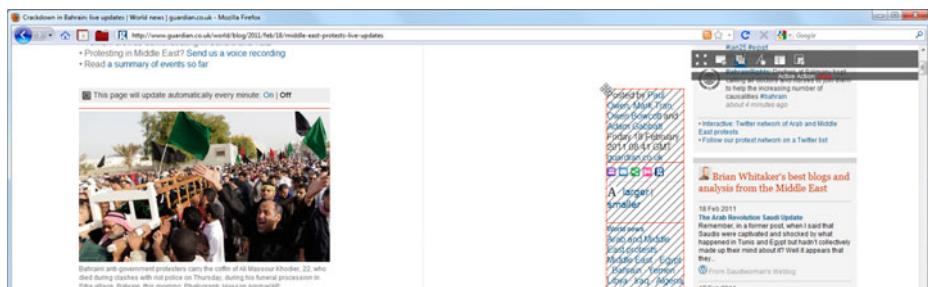
The adaptations carried out by users on different components of a web page, such as the header, navigation, content or footer, need to be reflected at the hypertext level where these are typically represented as a hierarchy of HTML elements or nodes associated with format and style defined in CSS. The two core operations supported by the toolkit allow web page elements to be repositioned and resized using drag-n-drop interaction as showcased in Fig. 2. This is technically supported by adjusting the CSS attributes related to the offset and width/height of elements while being dragged. When dropped, target elements can be positioned either relative to other elements or freely in the web page, which can be helpful when a nested design needs to be broken temporarily in the case that a number of grouped elements need to be realigned. Additionally, the margin between elements can be adjusted by adding new spacer elements to

Table 1. List of the adaptation operations supported by our toolkit with a focus on spatial factors of web site layout

Operation	Description
Move	Changes the position of target elements using drag-n-drop interaction. Moved elements can be docked at the left/top/right/bottom of the drop target, as well as float or be fixed at an absolute position in the web page.
Resize	Resizes target elements at their current position in the web page. When hovered, elements show a size grip in the lower right corner that can be dragged to change the width and/or height. Alternatively, the right side can be dragged to adjust only in the horizontal direction, or the bottom for vertical changes.
Spacer	Adds a new spacer element that will be docked to target elements and can be subject to other operations such as resize to increase the horizontal and/or vertical spacing between elements.
Hide	Hides target elements, or restores elements that were previously hidden using this operation.
Collapse	Substitutes target elements with a placeholder link that can be clicked to expand the original content.
Grow/Shrink Font	Increases or decreases the font size of target elements. Also line spacing can be controlled relative to the change of the font size, or using absolute values.
Single/Multi-column Layout	Controls the number of columns used for an element's layout so that content can be distributed horizontally and flow from one column to another in a flexible way.



(a) Resize operation for making appropriate use of the available width



(b) Move operation for realigning elements to fit the new dimensions

Fig. 2. Screenshots showing the adaptation toolkit and two example operations, namely (a) *resize* and (b) *move*, being performed directly in the browser.

improve on the layout where required. Users can also hide elements, or restore hidden ones. Alternatively, elements can be collapsed and replaced by a placeholder link that users can activate to unfold the substituted content. This feature is an example inspired by the mobile version of Wikipedia that automatically does this for sections of an article to first show only their headings. We believe that this kind of adaptation should be made available to a wider range of applications where it cannot be automated so easily because the content may not follow the wikitext conventions, but this could still be achieved manually with the help of users. Finally, the font size of text elements can be adjusted together with the line spacing and additional columns can be used for content layout, the number and size of which can be controlled using the single or multi-column layout operation.

The first set of operations in Table 1 primarily concern spatial factors of the design such as the size and positioning of content in the browser window. Users are provided with additional methods to adjust and balance the spacing between elements, which can be important to save some of the very limited space on mobile devices, or to make effective use of greater amounts of screen real estate on large displays. The means to collapse or even hide certain elements allows for simplifying complex layouts during the adaptation process at design-time, but also provides technical tools to optimise navigation and presentation of content at run-time according to the screen context. It is important to note that all operations can be combined with each other, except when elements are hidden, and generally apply to all web page elements. The last two have a specific focus on controlling text style and flow. This can be important to alleviate the problem of text appearing too small on large displays, or if the original font is too large on a small-screen device. Moreover, multiple columns can lead to a more effective use of the screen space in the horizontal direction, which can be essential for controlling the line length of paragraph elements that may otherwise get excessively long, especially in widescreen environments.

In two related projects, we have proposed a set of layout metrics [11] that can guide users in performing the adaptation as well as experimented with different adaptation techniques specifically in large-screen contexts [12]. The latter also expands more on the idea of using multi-column layout in wide-format viewing situations and explores ways of combining the resulting horizontal alignment of content with the dominant vertical scroll model used by the majority of web sites today.

5 Architecture

So far we have sketched the general idea of user-driven web site adaptations and presented a set of visual tools that facilitate the adaptation process. In this section, we present and discuss an architecture that can be integrated with existing web sites and applications with only minimal effort.

Figure 3 shows an extended version of the general client/server architecture behind a typical web application as well as processes (1) and (2) initiated by

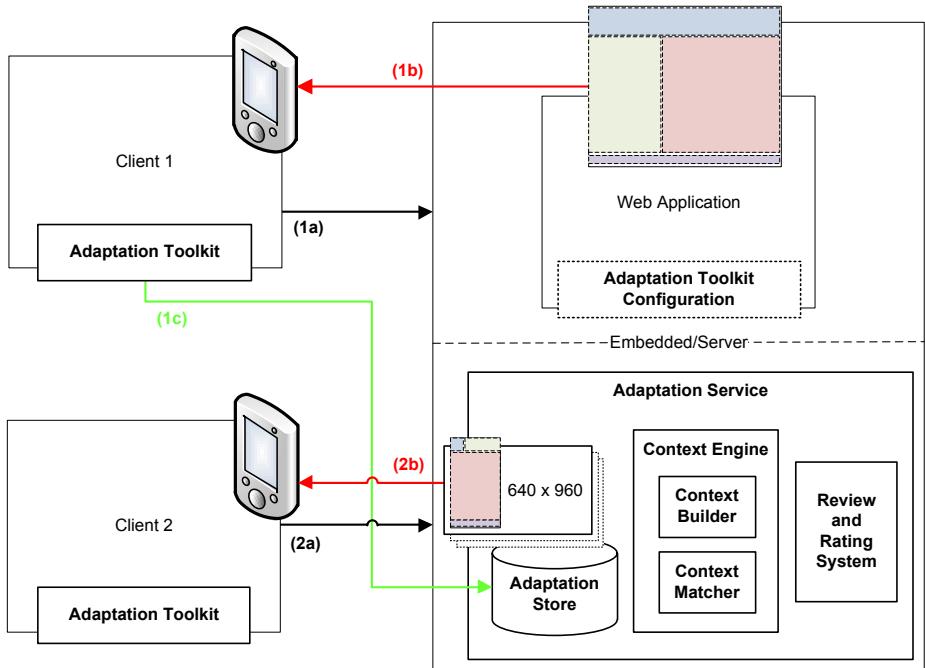


Fig. 3. Conceptual model of crowdsourcing architecture also showing the definition and deployment processes carried out by one client to the benefit of others

to separate clients. As for the extensions, both clients have the adaptation toolkit from the previous section installed and running in the browser. The web application can optionally define a configuration for the client toolkit in order for developers to control which and how web site elements can be adapted by users. Also located on the server-side is the adaptation service that comprises the following three components: (i) the adaptation store which is essentially an interface to an underlying database designed for the storage and retrieval of web site adaptations, (ii) the context engine which functions as a recommender system to rank and select the best-matching adaptations using the context builder and matcher, and (iii) the review and rating system for users to take influence on the recommendations made by the system. The dashed line labelled ‘Embedded/Server’ indicates that the architecture can vary at this point so that the adaptation service is either managed and provided as part of the particular web site, or administered by an independent service provider. Good examples of the latter scenario are the programmableweb.com or userscripts.org platforms mentioned earlier.

To give a concrete example of how the proposed architecture enables crowdsourced web adaptation, we show two clients using a mobile phone for accessing an example site that has initially been created with a standard resolution of 1024x768 in mind. When the first client accesses the application (1a), it will receive the content in its default layout (1b). This is however too large for the

small screen of the client device and therefore difficult to operate. As a countermeasure, client 1 uses our toolkit for adapting the web site to better fit the device resolution of 640x960. The example adaptations include shrinking of the header, repositioning and resizing of the navigation that was originally shown in the sidebar, maximising the main content area and adjusting the footer—all of which are supported by the proposed toolset. The client toolkit automatically synchronises with the adaptation store where it maintains a record with the defined adaptations for the current client context (1c), here simply represented by the screen resolution. Now when client 2 accesses the web site also using our toolkit (2a), it will download recommended adaptations and apply them (2b) so that the end-user automatically sees the mobile version that was created and shared by client 1.

6 Implementation

This section provides more detail on the enabling technologies for the adaptation toolkit and explains supported configuration options. Special attention will be given to the definition and deployment of web site adaptations as well as the quality control system. More documentation and the complete source code are available from our web site.³

Adaptation Toolkit. For the implementation of our toolkit, we use client-side JavaScript based on the popular jQuery framework⁴ to augment the target web site with a toolbar for invoking the different adaptation actions. If the script is not coupled with and distributed by the web application itself, as in the embedded mode of the architecture, we have also developed an extended version that additionally builds on Greasemonkey⁵—a browser extension for managing user scripts that can make on-the-fly changes to selected web pages as they are loaded. In this separate server setting, we do not constrain the domain and use the `@include http://*` statement to enable the adaptation toolkit for all web sites, but this default setting can be easily changed by users to work only with specific sites using the corresponding Greasemonkey features.

The adaptation operations offered by the toolkit are implemented as follows. The *move* and *resize* functions translate the corresponding *left* and *top* coordinates and adjust the *width* and *height* CSS properties. The *spacer* operation changes the CSS *padding* and *margin* properties of target elements to leave the indicated space blank. The *hide* function toggles the visibility of target elements by setting *display* to *none*. The *collapse* operation is realised using a combination of *display: none* for the original content and a CSS *:before* pseudo element in combination with the CSS *content* property to insert the placeholder link. The *grow/shrink font* operation directly translates to the corresponding CSS *font-size* and *line-height* parameters. Finally, *multi-column layout* is based on the

³ <http://dev.globis.ethz.ch/crowdadapt>

⁴ <http://jquery.com>

⁵ <http://www.greasespot.net/>

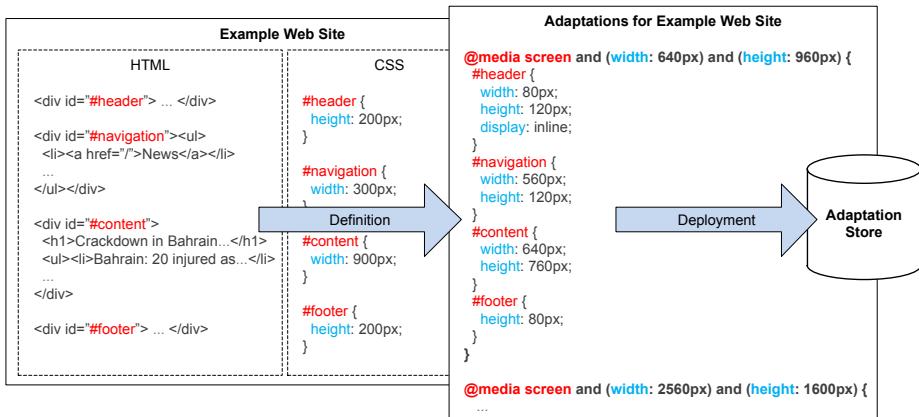


Fig. 4. Illustration of the adaptation process where adaptations are represented in CSS using media queries and stored for the example web site

new CSS3 multi-column module⁶ and therefore requires a modern browser that interprets modifications of the `column-count` and `column-width` properties.

The toolkit can further be configured to automatically disable links, text selection and embedded objects such as Flash animations or videos when performing the adaptation, not to interfere with the associated click handlers and other events that could get triggered accidentally. Also, target web sites can tell the toolkit to exclude certain web page elements from the adaptation. We support this by providing special CSS marker classes that can be added to HTML elements to prevent them from being moved, resized or hidden, for example. By default, adaptations can be performed on all non-restricted elements that have an `id`, `name` or `class`, as these can be easily accessed using jQuery and because it is the only feasible way to track the adaptations if not by the node index in the HTML document hierarchy. This also makes sense considering the fact that only important web page components are typically labelled for easy programmatic access or formatting using style blocks or separate CSS resources. Moreover, in case an element is adapted that also specifies a class, users are given the choice to apply the changes to all element instances of the selected class so as to learn from the adaptation of one page element as an example to also let it affect similar elements on the web page.

Adaptation Service. The adaptation store manages alterations of the layout in the form of CSS and CSS media queries as a result of the definition and deployment processes. To illustrate this, Fig. 4 shows a visualisation of the HTML and CSS of an example news article on the left and the adapted CSS on the right. In the original version of the web page, the header component specifies a fixed height and fills the full width of the viewport by default. Part of the adaptations performed by the first client from the previous scenario is the alteration of the header to require less space in both directions so that the navigation can

⁶ <http://www.w3.org/TR/css3-multicol/>

be aligned next to it. The results of the respective *move* and *resize* operations applied to the header and navigation components are decoded in separate CSS media queries. The styles defined here only apply if the device matches the indicated screen resolution. We therefore denote the resolution of the client device on which the adaptations were created originally. For the deployment, we support two modes so that adaptations can either be stored as a new, alternative *version* for the current client context, or as a *revision* to overwrite and replace a previous version. The underlying database model is fairly simple and just stores the adapted CSS together with the collected context parameters. In the case that the plugin was used, the URL of the original site is additionally tracked and stored in the database.

The context engine plays a bigger role for the retrieval of web site adaptations, but is also used for storage when the context builder creates a model from the context of the client that is performing the adaptations to associate the collected context information with user-defined adaptations. Our implementation evaluates the `navigator` and `screen` objects in JavaScript to collect the context information related to the browser and device, such as the user agent string, screen size, orientation and resolution. It also allows users to review and complement this information with aspects that cannot always be queried programmatically, such as the device type, model or name and supported input methods, e.g. mouse and keyboard, pen and/or touch. Clients can also specify user-related attributes for the context builder, e.g. their current location, the language preference and whether they are right or left-handed, although these values are not interpreted by default, but this is a way for users to hint at other context facets that might also be relevant for a particular use case and help to evolve both the context model and the adaptive behaviour of the system. The collected context information is then associated with the corresponding adaptations and stored in the database. At retrieval, the context matcher is responsible for scoring user-contributed adaptations and determining the best match when the site is accessed via the toolkit. The method used to evaluate similarity between the client contexts associated with adaptations in the database and the current client context is a combination of SQL and CSS media queries. SQL is used to filter stored adaptations by target web site and mostly by user-related context aspects not supported by media queries. The device-related matching process is however delegated to the client-side where it is performed directly in the browser using the returned CSS media queries.

Finally, the review and rating system complements the recommender system with user-driven means to control the scoring applied as part of the context matching process. We allow clients to preview and choose from a set of user contributions that achieved a reasonably high score computed by the context engine. We then build on the simple idea of ranking adaptations higher the more clients actually use them, for which we keep a server-side record of the number of client accesses together with the contexts of selected adaptations. The ratio of active to matching sets of adaptations then builds an additional factor for the matching process: it controls the order in which media queries are

cascaded by the toolkit so that only the adaptations with the highest scores are the ones that determine the final user interface.

7 Discussion

One of the key issues for any crowdsourcing model is the “cold-start problem” and so there has been a lot of research on motivational aspects. Results suggest that for users the willingness to contribute can be improved, for example, by using principles of social psychology [13] or by exploiting social connectedness [14], or indirectly, and for the user implicitly, through games [15]. Our approach attempts to primarily draw on the effect that users can see the interface improve directly through their actions. Similar to [16], user motivation can further be raised by letting them know how many other users with the same or similar devices can also benefit. This we can estimate by comparing the context against the history of client contexts used as part of our implementation.

Two aspects that often tend to be ignored in crowdsourcing approaches are security and privacy. As far as security is concerned, it is important to see that our approach focuses on crowdsourced *adaptation* rather than *augmentation* of web sites. Hence, it deals with adjusting the presentation rather than adding new content or functionality, which could be of potential concern, and therefore requires significantly less control compared to web augmentation [17]. Also privacy is not an issue since our focus is on the contributions of users rather than the users themselves. As a consequence, users can contribute anonymously while the sharing and targeting of user-defined adaptations is primarily based on stereotypical rather than personal aspects.

Another important aspect that we have not discussed so extensively in the paper is scalability when a crowd of users would really contribute with web site adaptations. This can have an impact on both the quality of adaptations and performance of the system. The suggested review and rating system could help maintain a high level of quality which could further be improved by giving more control to selected web site users, so-called trusted users, so that their approval or rejection has significant impact on the particular score and thus the overall ranking of user-defined adaptations. Another way to control quality is to limit which and how adaptations can be defined. The proposed adaptation operations therefore concern primarily those aspects of the design that are directly related to the viewing context, such as the size and position of elements, rather than style and colour. As for the performance, we argue that this is more a question of the implementation rather than the approach. Our experiments with the current implementation based on CSS and CSS media queries have shown a fairly high performance even when several hundred adaptations created for different client contexts were applied to an example web site. Even on mobile devices, modern browsers are very fast at parsing and executing the hypertext definitions and we can directly leverage this performance in our approach. In particular in a multi-user, multi-device scenario, we are convinced that the delegation of the adaptation process to the client-side will pay off as less of the more expensive

server-side computation is involved. The lightweight implementation also has some drawbacks. For example, we know it is better to remove hidden elements from the HTML before they are transferred to the client and to let the move operation also affect the hypertext specification, not to break the structure and flow of the document by relying on relative or absolute positioning via CSS independent of the document hierarchy. This means that docking a dragged element to a target should result in moving the corresponding element before or after the drop target's position in the HTML document. These are examples of adaptations that could be additionally implemented on the server-side to complement the pure CSS-based adaptation techniques with HTML pre-processing before the content is delivered to the client.

Finally, it must be mentioned that the adaptation toolkit itself also needs to be adaptive to support the adaptation process under any given device constraints. For example, while there is sufficient extra space to perform adaptations when working on a large device, the toolkit operations will also have to be feasible on small form-factor devices. We have started to support this by making the adaptation toolkit the subject of its own principles in that the crowd can also adapt the design and functionality of the toolkit. For the visual representation, we provide a special edit mode so that adaptation operations can also concern the toolkit components, while the source code is available for more experienced users to add new operations or refine existing behaviour programmatically.

8 Conclusions

We have presented a crowdsourcing approach to web site adaptation based on a lightweight extension of the common web application architecture and visual tools that facilitate the adaptation process. The main idea is to support developers in specifying web interfaces that can adapt to the range and increased diversity of web-enabled devices available today by also allowing users to create and share adaptations so that other owners of the same device and with similar preferences can directly benefit. The paper discussed the technical and design challenges for adopting this crowdsourcing model, in particular when it comes to quality control, and described a concrete implementation of the proposed concepts and mechanisms using adaptation techniques based on only CSS and CSS media queries. Although our experiments using the toolkit and architecture for adapting existing, real-world web sites are promising, more detailed evaluations of the visual toolkit from a user perspective and testing the implementation on a larger scale with many different user contributions are planned for the future. This will include carrying out user studies and collecting feedback when using the toolkit for access and adaptation on a range of different devices as well as experimenting with different sharing and ranking modes to improve user support. It will also be interesting to mine the data created through user participation as it is likely that user-driven adaptations generate valuable data for developers. The collected data can then inform the general design of a web site with respect to different use contexts so that patterns can be recognised and hopefully design deficiencies prevented before they get replicated.

Acknowledgements

We would like to thank Michael Grossniklaus and Stefania Leone for their valuable feedback on earlier drafts of this paper. This work was supported by the SNF under research grant 200021_121847.

References

1. Howe, J.: The Rise of Crowdsourcing. *Wired* 14(6) (2006)
2. Kazman, R., Chen, H.M.: The Metropolis Model: A New Logic for Development of Crowdsourced Systems. *CACM*, vol. 52(7) (2009)
3. Brusilovsky, P.: Methods and Techniques of Adaptive Hypermedia. *UMUAI* 6(2-3) (1996)
4. Ceri, S., Daniel, F., Matera, M., Facca, F.M.: Model-driven Development of Context-Aware Web Applications. *TOIT* 7(1) (2007)
5. Frăsincar, F., Houben, G.-J., Barna, P.: Hypermedia presentation generation in Hera. *IS* 35(1) (2010)
6. Daniel, F., Matera, M., Pozzi, G.: Managing Runtime Adaptivity through Active Rules: the Belleroonte Framework. *JWE* 7(3) (2008)
7. Niederhausen, M., van der Sluijs, K., Hidders, J., Leonardi, E., Houben, G.J., Meißner, K.: Harnessing the Power of Semantics-Based, Aspect-Oriented Adaptation for amacon. In: Proc. ICWE (2009)
8. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi- Target User Interfaces. In: IWC, vol. 15 (2003)
9. Paternò, F., Santoro, C., Mäntylä, J., Mori, G., Sansone, S.: Authoring pervasive multimodal user interfaces. *IJWET* 4(2) (2008)
10. Nebeling, M., Grossniklaus, M., Leone, S., Norrie, M.C.: Domain-specific language for context-aware web applications. In: Chen, L., Triantafillou, P., Suel, T. (eds.) *WISE 2010. LNCS*, vol. 6488, pp. 471–479. Springer, Heidelberg (2010)
11. Nebeling, M., Matulic, F., Norrie, M.C.: Metrics for the Evaluation of News Site Content Layout in Large-Screen Contexts. In: Proc. CHI (2011)
12. Streit, L.: Investigating Web Site Adaptation to Large Screens. Master's thesis, ETH Zurich, doi: 10.3929/ethz-a-006250434 (2010)
13. Harper, F.M., Li, S.X., Chen, Y., Konstan, J.A.: Social Comparisons to Motivate Contributions to an Online Community. In: de Kort, Y.A.W., IJsselsteijn, W.A., Midden, C., Eggen, B., Fogg, B.J. (eds.) *PERSUASIVE 2007. LNCS*, vol. 4744, pp. 148–159. Springer, Heidelberg (2007)
14. Bernstein, M.S., Tan, D.S., Smith, G., Czerwinski, M., Horvitz, E.: Personalization via Friendsourcing. *TOCHI* 17(2) (2010)
15. von Ahn, L., Dabbish, L.: Labeling Images with a Computer Game. In: Proc. CHI (2004)
16. Rashid, A.M., Ling, K.S., Tassone, R.D., Resnick, P., Kraut, R.E., Riedl, J.: Motivating Participation by Displaying the Value of Contribution. In: Proc. CHI (2006)
17. Arellano, C., Díaz, O., Iturrioz, J.: Crowdsourced Web Augmentation: A Security Model. In: Chen, L., Triantafillou, P., Suel, T. (eds.) *WISE 2010. LNCS*, vol. 6488, pp. 294–307. Springer, Heidelberg (2010)

A Layered Approach to Revisitation Prediction

George Papadakis^{1,2}, Ricardo Kawase², Eelco Herder², and Claudia Niederée²

¹ ICCS, National Technical University of Athens, Greece

gppapadis@mail.ntua.gr

² L3S Research Center, Leibniz University of Hanover, Germany

{surname}@l3s.de

Abstract. Web browser users return to Web pages for various reasons. Apart from pages visited due to backtracking, they typically have a number of favorite/important pages that they monitor or tasks that reoccur on an infrequent basis. In this paper, we introduce the architecture of a system that facilitates revisitations through the effective prediction of the next page request. It consists of three layers, each dealing with a specific aspect of revisit patterns: the first one estimates the value of each page by balancing the recency and the frequency of its requests; the second one captures the contextual regularities in users' navigational activity in order to promote related pages, and the third one dynamically adapts the page associations of the second layer to the constant drift in the interests of users. For each layer, we introduce several methods, and evaluate them over a large, real-world dataset. The outcomes of our experimental evaluation suggest a significant improvement over other methods typically used in this context.

1 Introduction

Revisitation is the act of accessing again a previously visited Web page. As such, it constitutes a major part of the entire Web activity: Web users usually have to handle repetitive but infrequent tasks, revisiting pages after a considerable amount of time [4]. This was verified by most past works that explored users' surfing behaviors; Herder [10], for instance, quantifies it to 50% of the overall Web traffic, while Cockburn and McKenzie [4] approximate it to 80%. As a result, individuals have been found to waste 15% of their overall browsing time in their effort to find information they have accessed in the past [18]. They can benefit, therefore, to a large extent from browser-based methods that predict and facilitate their next revisit request.

The most popular tools for client-side revisit are bookmarks [4,10] and search engines [18,19]. The former, though, had their popularity significantly declined in favor of the latter, as they involve serious managing and organizational problems: the size of bookmark collections constantly increases with time, thus reducing their usability [4]. Search engines, on the other hand, are becoming the dominant tool for supporting revisit, with about 40% of all queries pertaining to *re-finding*; that is, the process of using the same or a similar query to re-locate a previously visited Web page. However, the use of search engines

is impractical, as it requires the memorization of a usually hard-to-remember combination of keywords [11]. There is, therefore, a great need for new methods that predict and facilitate users' revisit activity.

In this paper, we introduce a system architecture that encompasses a set of methods aligned in three tiers. Each layer captures specific patterns in the navigational activity of a user, in order to effectively predict her next revisit: the first one, the *ranking layer*, comprises functions that rank visited resources according to their likelihood of being (re-)accessed in the immediate future. The second one, called *propagation layer*, enhances the ranking methods with techniques that encapsulate contextual patterns in the behavior of a user; that is, it identifies groups of pages visited together during the same session - in the same or different order - and boosts their ranking values accordingly. Finally, the third tier, the *drift layer*, conveys methods that adapt the patterns captured by the propagation layer to the changing nature of the interests of the user. On the whole, our framework constitutes a comprehensive method for revisit prediction, that covers all its aspects (i.e., frequency and recency of page requests, contextual patterns and concept drift), while being easy to implement and integrate into a user interface.

Special care has been taken to make our framework extensible, so that adding new methods or improving existing ones, in any of its three tiers, is a straightforward procedure. This is indeed ensured by the transparency of the strictly defined interfaces described in Section 3. We have also made public both the implementation and the data used in this paper under the *SUPRA*¹ project of SourceForge². Thus, we provide a common benchmark for new algorithms in this area, and encourage other researchers to experiment with our library and extend it with improved or novel techniques.

To summarize, the main contributions of this paper are the following:

- We introduce a layered architecture for a system that effectively addresses the next revisit prediction problem. It consists of three tiers, each tackling a specific aspect of the problem.
- We coin several methods for each layer, based on the navigational and the time patterns of individual user's activity.
- We evaluate the methods of our library through a thorough experimental study that involves a voluminous, real-world dataset. The results verify its superiority over well-established methods for this problem.

The rest of this paper is organized as follows: in Section 2 we discuss related work, while in Section 3 we formally define the problem we are tackling and elaborate on the architecture of our system. Section 4 analyses our thorough evaluation study, and Section 5 wraps up our work with final remarks and future plans.

¹ SUPRA stands for "SURfing Prediction fRAmework".

² See <http://sourceforge.net/projects/supraproject>.

2 Related Work

A problem more general than the *next revisit prediction* has been extensively studied in the literature: the *next page prediction* problem. The method that has prevailed in this field, at least in terms of popularity, is Association Rules Mining. In more detail, *association rules (AR)* effectively identify related resources without taking into account their order of appearance (e.g., pages that are typically visited together, in the same session, but not necessarily in the same order) [1,2]. This feature turns them ideal for recommending resources related to a particular site. Numerous works have investigated the functionality of different variations of AR [7,12,16]. For example, a recent work by Kazienko [12] explores indirect AR for Web recommendations, involving resources that are not “hardly” connected as in typical AR.

However, AR suffer from a variety of drawbacks: first, they rely on the most frequent patterns identified in the training set, thus misclassifying new patterns that are not included in it (e.g., patterns stemming from concept drift). Second, they fail to recommend rarely visited, and, thus, non-obvious and serendipitous items, since such resources never reach the minimum support limit³. Third, they disregard the order of itemsets, and cannot distinguish between different patterns that involve the same resources (i.e., an itemset $I_1 = \{1, 2, 3\}$ is treated equally with all its 6 permutations).

To overcome this last problem, *sequential patterns* have also been employed in the context of prediction methods. Among them, state-based methods, like Markov models, are particularly popular [20,6,3]. Sequence mining techniques constitute a variation of this approach, in the sense that they do not consider the strict order between items [2,15]. A comparison of these techniques with AR was conducted by Géry and Haddad [8], with the outcomes of their evaluation suggesting that Frequent Sequence Mining has the best performance. Nevertheless, all these methods still suffer from the inability to predict/recommend unseen items (i.e., not included in the training set, typically due to concept drift).

With the aim of introducing a prediction method that is equally effective with unseen data, Awad et al. [3] combined the Markov model with Support Vector Machines (SVM) under Dempster’s rule. Their experimental evaluation verified the superiority of their hybrid model over AR, especially when domain knowledge is incorporated into it. However, their method is quite impractical: it requires a different SVM classifier for each one of the available resources and, thus, entails an excessively high training time.

In the following sections, we propose a layered system architecture for revisit prediction that overcomes the shortcomings of existing works, while taking their advantages into account. To this end, the first layer incorporates techniques that estimate the value of visited pages from the frequency and the recency of their requests. The second layer, on the other hand, captures the

³ The problem of identifying rare but important associations has been tackled through the multiple minimum support method. This technique, however, has not yet been applied in the context of the next page prediction problem.

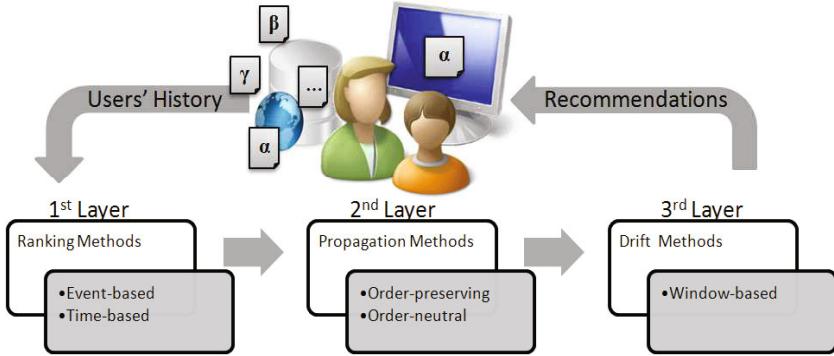


Fig. 1. The layered architecture of our revisitation prediction system

connections between pages visited during the same session, either by considering or by ignoring their order of access. Its novelty lies in its ability to identify new associations on-the-fly and to incorporate them dynamically into its data structure. To discard the connections that are outdated due to the drift in the interests of a user, we introduce another layer encompassing a window-based drift method; it re-adjusts the associations between pages after a certain period of time, so that they reflect the latest patterns in the user's activity.

3 Approach

The problem we are tackling in this paper consists of the task of identifying which Web page, among those visited by a specific user in the past, will be revisited in her next page request. More formally, we define it as follows:

Problem Statement. *Given the collection of Web pages, $P_u = \{p_1, p_2, \dots\}$, that have been visited by a user, u , during her past n page requests, $R_u = \{r_1, r_2, \dots, r_n\}$, order them accordingly, so that the ranking of the page p_i that she will revisit in her next request, r_{n+1} , is the highest possible.*

The above definition stresses that the goal is to facilitate the access to pages that have already been visited in the past, rather than trying to recommend not-visited but relevant ones. To serve this goal, we present a collection of methods that produce a ranking of all visited Web pages; the more likely a Web page is to be accessed in the next request, the higher its ranking. The ranked list of pages is updated after each page visit, and the higher the ranking of the subsequently accessed page, the better the prediction. This is in line with the intuition behind the ranking of search engines' results to keyword queries: users typically consult only the top 10 results, and the higher the ranking of the desired resource, the better the performance of the search engine [9].

Figure 1 depicts the architecture of our system, that encompasses three tiers of methods. The first one entails *ranking methods*, which estimate for each Web page the likelihood that it will be accessed in the next request. Their estimation

is derived from patterns in the surfing history of the underlying user, namely the recency and the frequency of accesses to each page. The second layer covers *propagation methods*; these are techniques that capture repetitiveness in the navigational activity of the underlying user and identify contextual associations between pages that are typically visited together (i.e., in the same session, but not necessarily in the same order). Depending on the degree of connectivity between the associated Web pages, their values (assigned by the ranking methods) are then propagated to each other. The third layer contains window-based *drift methods*, which adapt the associations encapsulated by the propagation methods to the volatile interests of the user. They employ a sliding time frame (e.g., of a day or a week) that periodically discards the connections that took place out of its borders. On the whole, these three layers provide a comprehensive framework that tackles all aspects of the revisit activity.

In the following, we present and analyze several techniques for each layer. Their implementation is already freely available through the *SUPRA* project of SourceForge. In this way, we encourage other researchers to experiment with them and to extend our library with new methods for every layer. Special care has been taken to make this a straightforward procedure, by providing clear guidelines through the formalization of the methods that are presented in the following sections. Any implementation complying with the minimal requirements for a ranking, a propagation and a drift method, as described in Definitions 1 to 8, can be easily integrated in our library. It is also worth noting that the real-world data employed in our experiments have also been publicly released through the *Web History Repository* project⁴, so that they can be used as a general benchmark for prediction algorithms, independently of our framework.

3.1 Ranking Methods

As mentioned above, the aim of a ranking method is to provide for each Web page a numerical estimation of the likelihood that it will be accessed in the next request. All pages are then sorted in descending order of their value, with the aim of placing the next revisited page to the highest possible ranking. After each page visit, the value of all pages changes, and the ranked list is updated. The reason is that the numerical estimation of each page is derived by contrasting the latest page visit with all (or part) of the past requests to that particular page; depending on the way the page's access history is handled, we distinguish two kinds of ranking functions: the event- and the time-based ones.

Time-based Ranking Methods. This family of ranking functions relies on the time the requests to a page occurred, in order to estimate its value. That is, the contribution of each request to the total value of the corresponding page depends on the actual time the respective page visits took place and the time that has elapsed ever since. Thus, the input of these methods principally comprises the request timestamps of each page:

⁴ See <http://webhistoryproject.blogspot.com>

Definition 1. Given the page requests R_u of a user u , the **request timestamps of a page** p_i , T_{p_i} , is the set of timestamps of those requests in R_u that pertain to p_i .

A time-based ranking method can be now defined as follows:

Definition 2. A **time-based ranking method** is a function that takes as input the request timestamps $T_{p_i} = \{t_1, t_2, \dots, t_k\}$ of a page p_i together with the time of the latest request, t_n , of the given user u , and produces as output a value for p_i , $v_{p_i} \in [0, 1]$, that is proportional to the likelihood that it will be accessed at the next page request, r_{n+1} (i.e., the closer v_{p_i} is to 1, the higher this likelihood).

In our system, we selected *Frecency (FR)* as representative of this kind of ranking methods. The reason is that it is integrated in one of the most popular Web browsers, namely Mozilla Firefox⁵. In essence, it places more emphasis on the frequency of the use of a Web page, and discounts only to some extent the influence of the very old visits. In more detail, the total ranking value of each page is equal to the sum of the values assigned to each of its requests; the value of a single page visit is called bonus and its size is proportional to its recency: requests occurring within the last four days take the highest bonus, whereas requests that are older than 90 days take the lowest one. In addition, FR considers the type of access, i.e., whether the URL of the page was typed, clicked upon or selected from the bookmarks collection. This is, however, out of the scope of our definition⁶. Note that to restrict the ranking values of Frecency in the interval $[0, 1]$, our implementation normalizes the value of each page with the globally largest page value.

Event-based Ranking Methods. In contrast with the previous category, event-based ranking methods interpret page visits as a sequence of events, and exclusively take into account their relative position. That is, they disregard the actual time of each request, and consider only the number of events that have elapsed since it occurred, in order to estimate its contribution to the total value of the corresponding page. Thus, this family of methods represents the access history of a page by the indices of the related requests:

Definition 3. Given the page requests R_u of a user u , the **request indices of a page** p_i , I_{p_i} , is the set of the serial numbers of those requests in R_u that pertain to p_i . The serial number of the chronologically first request is 1 and is incremented by 1 for each subsequent page visit.

Given this definition, an event-based ranking method is defined as follows:

Definition 4. An **event-based ranking method** is a function that takes as input the request indices $I_{p_i} = \{i_1, i_2, \dots, i_k\}$ of a page p_i together with the

⁵ See <http://www.mozilla.com/en-US/firefox>

⁶ See https://developer.mozilla.org/en/The_Places_frecency_algorithm for more details.

index of the latest request, i_n , of a user u , and produces as output the value of $p_i, v_{p_i} \in [0, 1]$, that is proportional to the likelihood that p_i will be accessed at the next page request, r_{n+1} (i.e., the closer v_{p_i} is to 1, the higher this likelihood).

As an illustration of the this kind of methods, we consider the *decay ranking model* that was introduced by Papadakis et al. in [14]. According to this model, the value of a Web page p_i after i_n visits is derived from the following formula:

$$DEC(p_i, I_{p_i}, i_n) = \sum_{j=1}^{|I_{p_i}|} d(i_j, i_n),$$

where $d(i_j, i_n)$ is a *decay function* that takes as an input the index i_j of a request to p_i together with the index of the current page vist, i_n , and gives as output the contribution of this request to the total value of p_i .

According to Cormode et al. [5], every *valid decay function* should satisfy the following properties:

1. $d(i_j, i_n) = 1$ when $i_j = i_n$
2. $0 \leq d(i_j, i_n) \leq 1 \quad \forall i_j \in [0, i_n]$
3. d is monotone non-increasing as n increases:

$$i'_n \geq i_n \rightarrow d(i_j, i'_n) \leq d(i_j, i_n) \quad \forall i_j \in [0, i_n].$$

Among the valid decay function families, the *Polynomial Decay (PD)* functions were found by Papadakis et al. [14] to outperform both the *exponential* and the *logarithmic ones*. The reason is that their smooth decay balances harmonically the recency and the frequency of page revisits; in contrast, exponential functions convey a steep decay that puts more emphasis on recency, whereas the logarithmic functions promote excessively frequency, due to their overly slow decay. The actual value of a polynomial decay function with exponent α for a page p_i at the $i_j - th$ request out of i_n , in total, accesses is given from the following formula:

$$d(i_j, i_n) = \frac{1}{1 + (i_n - i_j)^\alpha}. \quad (1)$$

The main difference between Polynomial Decay and Frecency, apart from the evidence they take into account, is the balance they achieve between frequency and recency. Frecency favors the former over the latter, thus constituting a mere improved version of the Most Frequently Used caching algorithm. On the other hand, Polynomial Decay achieves a better balance between these two metrics, while being more flexible, as well. In fact, it can be adapted to the behavioral patterns of the underlying user, employing the value of α in Formula 1 as a fine-tuning parameter; the larger its value ($1 << \alpha$), the higher the effect of recency on the overall value of a page (due to the steeper the decay of the contribution of a page request), and vice versa. Thus, Polynomial Decay can adjust its performance to different kinds of users.

3.2 Propagation Methods

Unlike ranking methods that produce an ordering of Web pages, propagation methods aim at capturing contextual information through the detection of patterns in the surfing activity of users. They identify those pages that are commonly visited within the same session and associate them with each other. The “links” created by these methods are then combined with a ranking method, so that the value of a Web page is propagated to its relevant ones. In this way, the higher the value of a Web page, the more the pages associated with it are boosted and the more their ranking is upgraded.

At the core of the associations between resources lies the notion of the session, which can be formally defined as follows:

Definition 5. A *session* S is the bag of all pages p_i visited by a user u in the same browser tab for a time period of up to 25.5 minutes ([8,17]), placed in chronological order, from the earliest to the latest one: $S = \{p_1, p_2, \dots, p_k\}$.

Based on Definition 5, propagation methods can be defined as follows:

Definition 6. A *propagation method* is a function that takes as input the last requested page, p_i , within a session, S , and defines appropriately the degree of connection between p_i and all the other pages visited during S . Hence, given two pages, X and Y , it returns a value, $v_{XY} \in [0, 1]$, that is proportional to the likelihood of Y being accessed immediately after X (i.e., the closer v_{XY} is to 1, the more likely this transition is).

In this work, we distinguish two families of propagation methods: the *order-preserving* ones, which take into account the order of the page requests within a session, and the *order-neutral* ones that disregard this order. For the former case, we consider transition matrices, whereas for the latter we examine association matrices.

Order-Preserving Propagation Methods. This category of propagation methods relies on the idea that Web pages are typically accessed in the same or similar order. Hence, given a session that contains a series of page requests ordered by time, they build the associations between pages according to this ordering: each page is connected only with the pages that precede it. To capture these transitions that form chronological patterns in the navigational activity of users, we introduce the transition matrix (**TM**).

In more detail, a TM is a two dimensional structure with its rows and columns representing the Web pages P visited so far by the given user u (Problem Statement); each cell $TM(x, y)$ expresses the number of times that a user visited page y after x . Given that a transition matrix respects the order of accesses within a session, it is not a symmetrical one: the value of $TM(x, y)$ is not necessarily equal to that of $TM(y, x)$. Moreover, its diagonal cells are all equal to 0: $\forall x \quad TM(x, x) = 0$. This is because there is no point in associating a page with itself; in case a requested Web page is revisited in the subsequent request,

A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
A	0	1	0	0	A	0	1	1	1	A	0	1	$\frac{1}{2}$	$\frac{1}{4}$	A	0	1	2	4
B	0	0	1	0	B	1	0	1	1	B	$\frac{1}{4}$	0	1	$\frac{1}{2}$	B	4	0	1	2
C	0	0	0	1	C	1	0	0	1	C	$\frac{1}{2}$	0	0	1	C	2	0	0	1
D	1	0	0	0	D	1	0	0	0	D	1	0	0	0	D	1	0	0	0

(a) (b) (c) (d) (e)

Fig. 2. The values of several types of the Transition and the Association Matrices after the last page request of the session: $S_1 : A \rightarrow B \rightarrow C \rightarrow D \rightarrow A$. a) corresponds to SM, b) to CM, c) to DM, d) to IM, and e) to AM.

its ranking will be high enough due to the value assigned to it by the ranking method and does not need to be boosted by the propagation method.

In the following, we introduce 4 different techniques for correlating Web pages according to the past navigational activity in the context of a transition matrix. They are intuitively illustrated through a simple walkthrough example. Given a set of 4 Web pages - A, B, C, D - and the following set of requests during a given session, $S_1 : A \rightarrow B \rightarrow C \rightarrow D \rightarrow A$, we can associate these pages in four different ways (taking into account the order of the accesses):

1. **Simple Connectivity Transition Matrix (SM).** For each transition $x \rightarrow y$ in the given session, only the value of the cell $TM(x, y)$ is incremented by one. The frequencies defined by this rule work exactly as a first-order Markov model. The rationale behind this approach is, thus, the expectation that requests tend to occur in the same strict order. Figure 2(a) depicts the values of the transition matrix according to this rule after the last transition of the given session, $D \rightarrow A$.
2. **Continuous Connectivity Transition Matrix (CM).** Each Web page visited within the current session is associated with all the previously accessed pages. In this way, it can effectively support requests that take place in a similar order, i.e., in the same direction, but not necessarily in the same sequence (e.g., $X \rightarrow Z \rightarrow Y$ and $X \rightarrow Y$). In our example, A is associated with all other Web pages after transition $D \rightarrow A$, incrementing the corresponding cells by one (Figure 2(b)).
3. **Decreasing Continuous Connectivity Transition Matrix (DM).** This strategy operates in a similar way as the previous one with the difference that it increments the cells of TM by a decay parameter representing the distance (i.e., number of transitions) that intervenes between the corresponding Web pages. Therefore, this form of transition matrix lies in the middle of SM and DM, supporting evenly requests that occur either in the same or in similar order. In our example, $TM(C, A)$ is incremented by $1/2$ after $D \rightarrow A$, since page C is two steps away from the page A . Figure 2(c) depicts the whole DM after the transition $D \rightarrow A$.
4. **Increasing Continuous Connectivity Transition Matrix (IM).** This is the inverted version of the previous strategy. Instead of decreasing the value added to $TM(x, y)$ according to the distance of pages x and y , it increases it

proportionally. Hence, it results in stronger connections between pages that are more distant, in an effort to identify the final destination of the given session. By boosting its value early enough, it can significantly restrict the number of irrelevant pages that the user visits before reaching its actual page of interest. The matrix produced by this rule after the last transition of our example is presented in Figure 2(d).

It is worth noting that SM is also used in Awad et al. [3], but its frequencies are merely used as features to a classification algorithm. In addition, CM is also employed in Parameswaran et al. [15] as the means of providing the frequencies of the probabilistic analysis that precedence mining involves.

Order-Neutral Propagation Methods. In contrast to the order preserving methods, the order-neutral ones are based on the idea that the temporal order of page visits within a session is not important; pages that are visited in the course of the same session should be equally connected with each other, regardless of their order and the number of transitions that intervene between them. The rationale behind this idea is that users may visit a group of pages X, Y, Z on a regular basis, but not necessarily in that order.

To model this idea, we introduce the association matrix (**AM**); similar to TM, AM is a matrix whose rows and columns are the Web pages P visited so far by the given user. The difference is that AM is built simply by associating all pages visited in a single session with each other; i.e., each Web page is connected not only with the pages preceding it, but also with those following it. Thus, an AM is always a symmetrical matrix with all its diagonal cells equal to 0 ($\forall x \text{ } AM(x, x) = 0$). Given the session S_1 of the above example, the resulting AM has all non-diagonal cells equal to one, as all resources were accessed during this session (Figure 2(e)).

Combining Ranking with Propagation Methods. To combine a ranking method with one of the propagation techniques, we employ a simple, linear scheme: following the i_n -th page request, the value of all pages is (re)computed, according to the selected ranking method. Then, for each non-zero cell of the matrix at hand ($TM(x, y)$ or $AM(x, y)$), we increment the value assigned to page y by the ranking method, v_y , as follows:

$$v_y+ = p(x \rightarrow y) \cdot v_x, \text{ where}$$

- $p(x \rightarrow y)$ is the transition probability from page x to page y , estimated by $p(x \rightarrow y) = \frac{TM(x, y)}{\sum_i^{i_n} TM(x, i)}$ (or $p(x \rightarrow y) = \frac{AM(x, y)}{\sum_i^{i_n} AM(x, i)}$), and
- v_x is the value of x estimated by the ranking method.

3.3 Drift Methods

Unlike the ranking methods, the propagation ones encompass no inherent support for drift in the focus of user's interests: the connections stored in their data

structure (i.e., matrix) remain static, and, thus, cannot adapt to the constantly changing habits and interests of users. In the literature, two main approaches that support *concept drift* have been proposed: first, the decay functions, like the polynomial one, and, second, the window-based methods [13]. The latter take their name from the sliding window they employ in order to keep the most recent evidence and ignore the rest.

To enable the dynamic nature of the propagation methods, we introduce in our system a third layer that consists of a *window-based drift method*, operating on the data structure of the second layer. Depending on the way the size of the window is specified, we distinguish between *event-based* and *time-based* drift methods; the former define the window with respect to the size of a batch of requests, whereas the latter with respect to a period of time. More formally, the time-based drift methods are defined as follows:

Definition 7. *Given the page requests R_u of a user u , the matrix m of a propagation method and a time period t , a **time-based drift method** updates the connections stored in m so that they reflect the page requests of R_u that occurred in the latest t temporal units (e.g., days or weeks).*

Similarly, the event-based drift methods are defined as follows:

Definition 8. *Given the page requests R_u of a user u , the matrix m of a propagation method and a number of requests n , an **event-based drift method** updates the connections stored in m so that they reflect the page requests of R_u that occurred in the latest n page requests.*

Due to the temporal, periodic patterns we identified in the large, real-world data set we have at our disposal, we considered only time-based drift methods. Thus, in our experimental study we examine the **Day**-, the **Week**- and the **Month-model**. As their name suggests, they update the matrix of the underlying propagation method so that it maintains the associations of the last day, the last week and the last month, respectively.

4 Evaluation

Data Set. To thoroughly evaluate our approach, we employed a real-world, voluminous data set that was gathered through the Web History Repository project. It comprises the navigational activity of 200 users, logged in the time period between 30/09/2010 and 11/01/2011. In total, more than 580,000 page requests were recorded. They are not, though, evenly distributed over the participants; characteristically, there are 100 users with less than 1,000 requests (with a minimum of 200), and 18 users with more than 10,000 requests (with a maximum of 16,570). The distribution of the logging period per user varies greatly, as well, ranging from 1 to 278 days. On average, though, each user issued almost 3,000 page requests, in a time period of 38 days. One third of them constituted a revisit, thus producing a revisit rate that is a bit lower than the estimation

Table 1. Technical characteristics of the WHR data set we employed

Users	200	Av. Requests per User	2,914
Page Requests	582,853	Av. Web Pages per User	1,905
Web Pages	381,066	Av. Revisits per User	1,009
Sessions	91,300	Av. Sessions per User	457
Revisits	201,787	Av. Requests per Session	126.40
Revisitation Rate	34.62%	Av. Days per User	37.72

of Herder et al. [10] in 2005. Regarding the demographics (e.g., age and sex) of the participants, we do not have any relevant data at our disposal, since the volunteers contributed their navigational history anonymously. Note also that the sessions in the data set are set transparently by the browser, not necessarily according to the time criterion of Definition 5. The technical characteristics of our data set are summarized in Table 1.

Setup. In the course of our experimental evaluation, we simulated the navigational activity of each user independently of that of the others: her page requests were sorted in ascending order of time, and the simulation proceeded by one request at a time, starting from the earliest and moving to the latest one. A ranked list of the so-far-visited pages was maintained per user, and, after each page request, the ranking of all pages was updated, according to our prediction methods; if the next page request was not a revisit, the new page was added to the list. Otherwise, the position of the corresponding Web resource was recorded. Based on the ranking positions we collected, we evaluate the performance of the prediction algorithms in terms of the following metrics:

(i) **Success Rate at 1 (S@1)** denotes the portion of revisitations that pertained to the top ranked Web page. The higher this percentage, the better the performance of the prediction method.

(ii) **Success Rate at 10 (S@10)** stands for the percentage of revisit requests pertaining to a page that is ranked in some of the top 10 positions. Similar to S@1, the higher its value, the better the performance of the method. This metric expresses the actual usability of a prediction algorithm, as users typically have a look only at the first 10 pages presented to them, just like they do with Web search engine results [9].

(iii) **Average Ranking Position (ARP)** represents the place a revisited page is found on average in the ranking list of the method at hand. Thus, it provides an estimation of the overall performance of a prediction algorithm, considering the ranking position of all revisitations, and not just the top ranked ones. The lower its value, the better the performance of the prediction method.

On the whole, the combination of these three metrics provides a comprehensive estimation of the effectiveness of a prediction algorithm: it considers both its practical recommendations (S@1, S@10) and its performance over all revisitations (ARP).

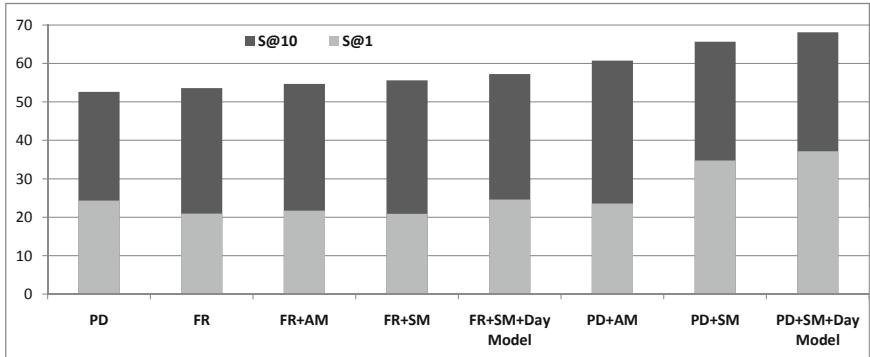


Fig. 3. Performance in percentage (%) with respect to S@1 and S@10 for the selected prediction methods. The methods are placed in ascending order of S@10 from left to right, with the right-most one achieving the optimal performance.

Results Analysis. As baseline approaches for the evaluation of our system, we consider the individual ranking methods of Section 3, i.e., FR and PD; they have already been proposed in the literature, and the former actually constitutes the state-of-the-art method, as it is integrated in a popular browser (Mozilla Firefox), and is widely used. The goal is, thus, to verify that combining existing ranking methods with the propagation and the drift ones enhances their performance to a significant extent and results in more accurate predictions.

In total, our framework accommodates (2 ranking methods \times 5 propagation methods \times 3 drift methods =)30 combinations of prediction methods. Due to lack of space and for the sake of readability, we will consider only 6 of them, in addition to the baseline ones: the best of combination of each ranking method with an order-neutral and an order-preserving propagation method, as well as the best combination of ranking and propagation methods with the day-model drift method. The criterion for choosing the best combination of ranking and propagation methods was their performance for S@10, the most indicative metric for the usability of a revisit prediction method. In this aspect, both ranking functions maximized their performance when combined with SM. On the other hand, the selection of the drift method was determined by the characteristics of our data set, which does not cover the long-term navigational activity of the participants (apart from a couple of users); rather, it contains their short- or their mid-term activity (i.e., few days and weeks, respectively). Nevertheless, our evaluation enables us to examine the contribution of each layer to the overall performance, independently of that of the others⁷.

The results of the evaluation for the first two metrics are presented in Figure 3, while Figure 4 depicts the performance of the selected methods for ARP. In

⁷ Note that there is not point of examining the performance of ranking methods in combination with the drift ones alone, since the latter apply only to the data structure of the propagation methods.

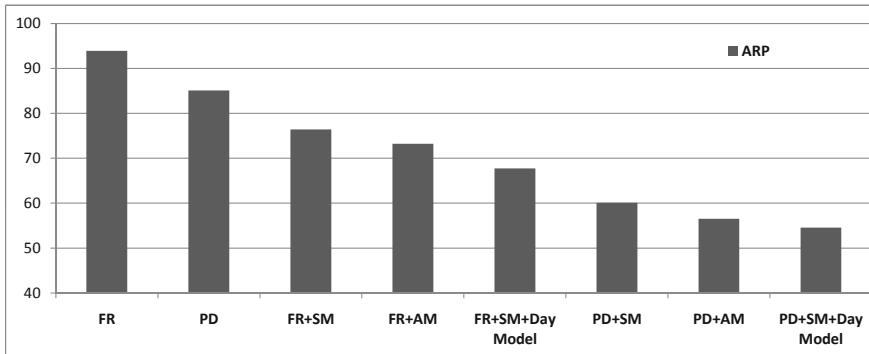


Fig. 4. Performance with respect to ARP for the selected prediction methods. The methods are placed in ascending order of performance from left to right, with the right-most one achieving the optimal performance.

both cases, the methods are ordered from left to right in ascending order of performance, so that the right-most method exhibits the best performance.

In the case of S@1 and S@10, we can easily notice that the performance of ranking methods is substantially improved by both the other levels of the system. Actually, the best performances are achieved when all three layers are employed in conjunction. This is particularly true for PD and its combinations with AM, SM and the Day Model, with the complete method (PD+SM+Day Model) improving PD by 52.84% and 29.41% with respect to the S@1 and S@10, respectively. FR, on the other hand, is improved to a lower extent by the three-layered model by 6.8% and 17.41%, respectively. All improvements of the second layer over the first and the third over the two other layers, were found to be statistically significant ($p < 0.05$), with the exception of FR+AM over FR.

Regarding ARP, we notice the same patterns: the more layers are employed, the better the overall performance of our system. It is worth noting that the degree of improvement for FR is much higher in this case: it ranges from 18.66% for FR+SM to 27.87% for FR+SM+Day Model. For PD, the improvements fluctuate between 29.38% for PD+SM and 35.88% PD+SM+Day Model. Again, all improvements of one layer over the underlying ones were found to be statistically significant ($p < 0.05$).

It is worth noting that the order-preserving propagation methods outperform the order-neutral ones for S@1 and S@10, while having a lower performance with respect to ARP. The reason is that, unlike SM, AM does not identify the most relevant page(s) to the currently accessed one; rather, it uniformly associates each page with all other pages visited during the same sessions. Thus, AM evenly distributes the value of the most recently visited page among all relevant pages, leading to higher ARP, whereas SM merely boosts the value and the ranking of the most likely next pages. The success rate gets, therefore, substantially higher, but ARP is not improved at a lower rate.

5 Conclusions

In this paper, we presented a layered architecture for a system that facilitates the revisit activity of users, through accurate predictions. It consists of three tiers, each addressing a particular aspect of this phenomenon: the recency and frequency of patterns of revisitations, their contextual patterns as well as the ever-changing interests of a user. Our thorough experimental evaluation verified that each layer conveys significant improvements over the prevalent method for this task (i.e., Mozilla Firefox's Frecency). On the whole, our system predicts the next revisited page in 37% of the cases, while its top-10 recommendations contain the desired page in 68% of the cases. Given that different users receive the optimal recommendations for different methods, in the future we plan to investigate ways of inferring a priori the optimal combination of methods for each user. In addition, we intend to examine whether our system is applicable in the context of server-side recommendations, as well (e.g., in the case of the intranet of a large company).

Acknowledgments

This research has been co-funded by the European Commission within the eContentplus targeted project OpenScout, grant ECP 2008 EDU 428016.

References

1. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: SIGMOD Conference, pp. 207–216 (1993)
2. Agrawal, R., Srikant, R.: Mining sequential patterns. In: ICDE, pp. 3–14 (1995)
3. Awad, M., Khan, L., Thuraisingham, B.M.: Predicting www surfing using multiple evidence combination. VLDB J. 17(3), 401–417 (2008)
4. Cockburn, A., McKenzie, B.J.: What do web users do? an empirical analysis of web use. Int. J. Hum.-Comput. Stud. 54(6), 903–922 (2001)
5. Cormode, G., Shkapenyuk, V., Srivastava, D., Xu, B.: Forward decay: A practical time decay model for streaming systems. In: ICDE, pp. 138–149 (2009)
6. Deshpande, M., Karypis, G.: Selective markov models for predicting web page accesses. ACM Trans. Internet Techn. 4(2), 163–184 (2004)
7. Fu, X., Budzik, J., Hammond, K.J.: Mining navigation history for recommendation. In: IUI, pp. 106–112 (2000)
8. Géry, M., Haddad, M.H.: Evaluation of web usage mining approaches for user's next request prediction. In: WIDM, pp. 74–81 (2003)
9. Hawking, D., Craswell, N., Bailey, P., Griffiths, K.: Measuring search engine quality. Inf. Retr. 4(1), 33–59 (2001)
10. Herder, E.: Characterizations of user web revisit behavior. In: LWA, pp. 32–37 (2005)
11. Kawase, R., Papadakis, G., Herder, E., Nejdl, W.: The impact of bookmarks and annotations on refinding information. In: HT, pp. 29–34 (2010)
12. Kazienko, P.: Mining indirect association rules for web recommendation. Applied Mathematics and Computer Science 19(1), 165–186 (2009)

13. Koychev, I., Schwab, I.: Adaptation to drifting user's interests. In: ECML Workshop: Machine Learning in New Information Age, Citeseer, pp. 39–46 (2000)
14. Papadakis, G., Niederee, C., Nejdl, W.: Decay-based ranking for social application content. In: WEBIST, pp. 276–282 (2010)
15. Parameswaran, A.G., Koutrika, G., Bercovitz, B., Garcia-Molina, H.: Recsplorer: recommendation algorithms based on precedence mining. In: SIGMOD, pp. 87–98 (2010)
16. Sandvig, J.J., Mobasher, B., Burke, R.: Robustness of collaborative recommendation based on association rule mining. In: RecSys, pp. 105–112 (2007)
17. Tauscher, L., Greenberg, S.: How people revisit web pages: empirical findings and implications for the design of history systems. Int. J. Hum.-Comput. Stud. 47(1), 97–137 (1997)
18. Teevan, J., Adar, E., Jones, R., Potts, M.A.S.: Information re-retrieval: repeat queries in yahoo's logs. In: SIGIR, pp. 151–158 (2007)
19. Tyler, S.K., Teevan, J.: Large scale query log analysis of re-finding. In: WSDM, pp. 191–200 (2010)
20. Yao, Y., Shi, L., Wang, Z.: A markov prediction model based on page hierarchical clustering. Int. J. Distrib. Sen. Netw. 5(1), 89–89 (2009)

Improving the Exploration of Tag Spaces Using Automated Tag Clustering

Joni Radelaar, Aart-Jan Boor, Damir Vandic,
Jan-Willem van Dam, Frederik Hogenboom, and Flavius Frasincar

Erasmus University Rotterdam
PO Box 1738, NL-3000
Rotterdam, The Netherlands
`joni@radelaar.nl, {aartjan.boor,jwvdam}@gmail.com,`
`{vandic,fhogenboom,frasincar}@ese.eur.nl`

Abstract. Due to the increasing popularity of tagging, it is important to overcome challenges resulting from the free nature of tagging, such as the use of synonyms, homonyms, syntactic variations, etc. The Semantic Tag Clustering Search (STCS) framework deals with these challenges by detecting syntactic variations of tags and by clustering semantically related tags. We evaluate our framework using Flickr data from 2009 and compare the STCS framework to two previously introduced tag clustering techniques. We conclude that our framework performs significantly better in terms of cluster precision compared to one method and has a better average precision compared to the other method.

Keywords: Tagging, syntactic clustering, semantic clustering, tag disambiguation.

1 Introduction

On today's World Wide Web, it is becoming increasingly popular to use tags for the purpose of describing resources. Tagging allows users to annotate a resource, such as a video, photo, or Web page, with a keyword or tag of their own choice. Because there are no restrictions on the tags that can be used, tags provide a flexible way of describing resources. However, because of the unstructured nature of tagging, there are some problems associated with retrieving resources using tag-based search engines. These problems are often caused by different tags having the same or closely related meaning. This can be the result of the use of synonyms, but it could also be caused by syntactic variations. Examples of syntactic variations are misspellings or the use of the plural or singular form of a specific word. Users may also use different levels of specificity while describing a resource, which is identified as the basic level variation problem by Golder and Huberman [9]. For example, one user might tag a picture of a cat as "animal" (not very specific), while another user would use "persian" (very specific). The usage of homonyms, i.e., words with multiple unrelated meanings, is another problem associated with tagging.

The problems described above can lead to undesirable results when searching for resources using tags. For example if a user is looking for a picture using “cat” as a keyword, he or she would most likely also be interested in pictures which are tagged with “cats” (syntactic variation), “persian” (more specific, semantically related term), “kitty” (synonym), and “kittie” (misspelling of kitty).

One way to deal with these problems is to create clusters of syntactically and semantically related tags. Creating syntactic clusters involves the grouping of tags that are syntactic variations of each other into separate groups or clusters. Search algorithms can then use these clusters to improve the quality of a search query. For example, when a user enters a tag as a search query, the search algorithm could also add tags to the query that are in the same cluster as the tag that was entered. Creating semantically related clusters involves grouping tags that are semantically related, e.g., “sanfrancisco” and “goldengate”. Tags occurring in multiple semantic clusters can be used to identify tags with multiple meanings. If a tag occurs in multiple clusters it most likely also has multiple meanings, e.g., “turkey” can refer to both the country and the animal.

As a solution to the previous problem, we define the Semantic Tag Clustering Search (STCS) framework, which consists of two parts. The first part deals with syntactic variations, whereas the second part is concerned with deriving semantic clusters. We implement and evaluate the use of the Levenshtein similarity measure [13] and a combination of the Levenshtein similarity and the cosine similarity measure, as similarity measures for syntactically related tags. For identifying semantic clusters we implement and evaluate the semantic clustering algorithm proposed by Specia and Motta [21] and a clustering algorithm proposed by Lancichinetti et al. [11]. Additionally, we propose a modification to the Specia and Motta approach to improve the results. We perform a thorough evaluation of the used clustering methods that goes beyond previous evaluations in extent.

The contribution of this paper stems from several aspects. First, although several clustering techniques for clustering tags have already been proposed [3,21,24], the evaluation of these techniques is done using relatively small data sets with a small number of resources. In this paper, we evaluate different syntactic and semantic clustering techniques using a larger data set than previously reported in literature. In this way we aim to analyze the performance of our algorithms more accurately on high volume data, gathered from Flickr [20]. Second, the proposed algorithm for syntactic clustering addresses the issue of identifying syntactic variations among short tags. Third, for the semantic clustering we identify the issues with currently available tag clustering algorithms and propose solutions for them. We have published previous work on STCS in [23]. Compared to this early work, in this paper we provide more details on the used algorithms, use a significantly larger data set for the experiments, and perform a more thorough evaluation.

The rest of this paper is organized as follows. Section 2 discusses related work. Subsequently, in Section 3 and 4 we give an overview of the design and

implementation of our STCS framework. Section 5 elaborates on the evaluation of our experimental results and Section 6 concludes the paper.

2 Related Work

This section discusses related work on several key aspects of our methodology. Firstly, Subsection 2.1 presents tag clustering methods. Then, Subsection 2.2 elaborates on similarity measures, and finally, Subsection 2.3 introduces some related work on cluster evaluation.

2.1 Tag Clustering

Echarte et al. [7] discuss the problem of syntactic variations in folksonomies. They propose the utilization of pattern matching techniques to identify syntactic variations of tags. They evaluate the performance of the Levenshtein and Hamming distances using the 10,000 most popular tags and 1,577,198 annotations from CiteULike. Results show that the Levenshtein measure provides the best overall performance. However, both techniques do not perform well with tags shorter than 4 characters.

Specia and Motta [21] propose a method for building semantically related clusters of tags using a non-hierarchical clustering technique based on the co-occurrence of tags. They also explore the relationships between pairs of within-cluster tags. The authors perform a statistical analysis of the tag space in order to identify clusters of possibly related tags. Clustering is based on the cosine similarity among tags given by their co-occurrence vectors. Before creating the clusters, Specia and Motta merge morphologically similar tags using the normalized Levenshtein distance measure. The authors manually evaluated the results based on 49,087 distinct resources and 17,956 distinct tags from Flickr and found that the clustering approach results in meaningful groups of tags corresponding to concepts in ontologies.

Begelman et al. [3] propose to build a directed graph of tags with an edge between two vertices (tags) when there is a (strong) relation. The weight of the edge is based on the co-occurrence of the connected tags. In order to partition the set of tags into groups of semantically-related tags, their recursive algorithm uses spectral bisection to split the graph into two clusters. It then evaluates the split using the modularity function, which was introduced by Newman and Girvan [17]. The modularity function provides a measure of the quality of a particular division of a network. Begelman et al. applied their clustering algorithm to a data set containing 200,000 resources and 30,000 tags.

Yueng et al. [24] also use a graph-based clustering algorithm, where the modularity function is used to evaluate the quality of a division. However, unlike Begelman et al., the authors consider different network representations of tags and documents, e.g., networks based on users, co-occurrence of tags, and context of tags using cosine similarity, and discuss the effects of these various representations on the resulting clusters of semantically related tags. For their

clustering experiments, a small data set is gathered from Delicious, containing 20 manually selected tags representing two or more concepts, complemented by randomly selecting 30 tags from the 100 most popular tags, with each tag having about 500 images. The authors find that networks based on tag context similarity capture the most concepts. With these networks the cosine similarity is used to perform a comparison of the context in which two tags are used, as reflected by the tag co-occurrence vectors of the tags.

Lancichinetti et al. [11] present a method that uncovers the hierarchical and overlapping community structure of complex networks. Their algorithm uses a newly defined fitness function that determines the quality of a cover. This function is used to discover the natural community of each node in a graph by optimizing the fitness function using local iterative searching. The authors evaluate their algorithm on artificial networks [2] that are known to have a built-in community structure. Their results show that their algorithm is successful in identifying these communities.

2.2 Similarity Measures

There are many similarity measures available for use in clustering algorithms. Cattuto et al. [4] analyze a variety of these using a Delicious data set containing the 10,000 most popular tags, by comparing the relations established through the use of different similarity measures to WordNet [8] synsets. Cosine similarity turns out to be the best similarity measure for detecting synonyms, while FolkRank and co-occurrence appear to be more useful for detecting various other semantic relations.

Markines et al. [15] evaluate the matching similarity, overlap similarity, Jaccard similarity, Dice coefficient, cosine similarity, and mutual information measures using a more systematic approach. They investigate the performance of these measures by generating several two-dimensional views on the tripartite folksonomy of BibSonomy [10] using aggregated data from 128,500 resources, 1,921 users, and 58,753 tags. Unlike BibSonomy, Flickr only allows one user to annotate a resource. Therefore, for a Flickr data set, only projection aggregation is useful. The result of projection aggregation can be considered as a matrix with binary elements $w_{rt} \in \{0, 1\}$, where rows correspond to resources and columns corresponds to tags. Given a resource and a tag, a 0 in this matrix element means that no user associated that resource with that tag, whereas a 1 means that at least one user has performed the indicated association. All similarity measures can then be derived directly from this information. The usefulness of the various measures as tag-tag similarity measures is evaluated using Kendall's τ correlations between the similarity vectors generated by the various measures and the reference similarity vector provided by a WordNet grounding measure. Mutual information proved to be the best similarity measure when using projection aggregation. When compared to each other the remaining similarity measures have the same performance. Unfortunately, mutual information is a computationally intensive measure, which makes its use unfeasible for large data sets.

2.3 Cluster Evaluation

Several measures exist to analyze clusters. Larsen and Aone [12] describe the precision measure. Average precision is defined as

$$\text{AvgPrec}(\Omega, C) = \frac{1}{|\Omega|} \sum_{w_k \in \Omega} \frac{\max_{c_j \in C} |\omega_k \cap c_j|}{|w_k|}, \quad (1)$$

where $\Omega = \{\omega_1, \omega_2, \dots, \omega_k\}$ is the set of tag clusters and $C = \{c_1, c_2, \dots, c_j\}$ is the set of tag classes. We interpret ω_k and c_j as a set of tags, where ω_k is denoting a tag cluster and c_j a tag class.

Manning et al. [14] describe the purity measure to evaluate clusters. They define purity as

$$\text{Purity}(\Omega, C) = \frac{1}{N} \sum_{w_k \in \Omega} \max_{c_j \in C} |\omega_k \cap c_j|, \quad (2)$$

where Ω and C are the same as in the previous equation and N is the total number of tags.

Delling et al. [6] propose the density measure, which is a trade-off between intra-cluster density and inter-cluster sparsity to evaluate a specific clustering. Let us assume that G is an undirected and unweighted graph with n nodes and m edges. A partitioning of the nodes into several clusters c is called a clustering C of a graph. The edges between nodes of the same cluster are called *intra-cluster edges* and the edges between nodes of different clusters are called *inter-cluster edges*. The density of clustering C is then defined as:

$$\text{Density}(C) = \frac{1}{2} \text{ Intra-cluster-density}(C) + \frac{1}{2} \text{ Inter-cluster-sparsity}(C), \quad (3)$$

where

$$\text{Intra-cluster-density}(C) = \frac{1}{|C|} \sum_{c \in C} \frac{\# \text{ intra-cluster edges } c}{\binom{|c|}{2}}, \quad (4)$$

and

$$\text{Inter-cluster-sparsity}(C) = 1 - \frac{\# \text{ inter-cluster edges}}{\binom{n}{2} - \sum_{c \in C} \binom{|c|}{2}}. \quad (5)$$

3 Framework Design

This section introduces the Semantic Tag Clustering Search framework (STCS) framework, which addresses the syntactic and semantic issues in tagging systems. The framework consists of two layers. In the first layer, syntactic variations (e.g., misspellings, morphological variations, etc.) of tags are eliminated by clustering the tags that are syntactic variations of each other and merging them into a single tag. In the second layer, the framework deals with the problem of identifying semantically related tags. This section continues with a problem definition in Subsection 3.1, a discussion of the similarity measures used in Subsection 3.2, and a more detailed elaboration of the framework in Subsection 3.3.

3.1 Problem Definition

We now give a formal problem definition, for which we follow the formulation given in [16]. The data set which is used as input for the framework is defined as a tuple $D = \{U, T, P, r\}$, where U , T , and P are the finite sets of users, tags, and pictures, respectively. The ternary relationship $r \subseteq U \times T \times P$ defines the initial annotations of the users.

Removing Syntactic Variations. In order to effectively find semantically related tags, we first remove syntactic variations of tags from the data set. Syntactic variations usually are misspellings of words but may also include translations of tags in other languages, or morphological variations. To remove these syntactic variations we create a set $T' \subset \mathcal{P}(T)$ in which each element of the set T' is a cluster containing all tags that are syntactic variations of each other. Each tag can only appear in one cluster. To determine the tag to be used as cluster label, we define m' , which is the bijective function that indicates a label for each $x \in T'$, $m' : T' \rightarrow L$. For each $l \in L$ and some $x \in T'$, $l \in x$ holds, such that $m'(x) = l$, thus, l is one of the tags that labels the cluster x .

Finding Semantically Related Tags. In our framework, we aim to find semantically related tags by creating a set T'' containing semantic clusters of elements $l \in L$. This denotes that we disregard the syntactic variations in the semantic clusterings by only clustering tags that are labels of syntactic clusters. An example of a semantic cluster is $\{\text{"nyc"}, \text{"newyork"}, \text{"manhattan"}\}$. A tag should be able to be part of multiple clusters, each with a different meaning.

3.2 Similarity Measures

Based on the results of the discussed related work, we apply two similarity measures within the STCS framework. In order to determine tag similarity, we employ the Levenshtein distance measure and the cosine similarity. Related work showed that the Levenshtein measure performed better in detecting syntactic variations than the Hamming distance measure [7]. However for short tags the Levenshtein measure does not perform well. In order to cope with this problem we have combined the Levenshtein distance with the cosine similarity. The Levenshtein distance is a measure for the amount of typographic difference between two strings, also called edit distance. It is defined as the minimum number of operations needed to transform one string into the other. An operation can be an insertion, deletion, or substitution of a single character. We call this distance the *absolute* Levenshtein distance. We denote it by alv_{ij} , which is the absolute Levenshtein distance between tag i and j . Our framework needs to deal with different tag lengths so we used the *normalized* Levenshtein similarity, which is a measure that is relative to the tag length. The normalized Levenshtein similarity between tag i and j , denoted by lv_{ij} , is defined as follows:

$$lv_{ij} = 1 - \frac{alv_{ij}}{\max(\text{length}(t_i), \text{length}(t_j))}. \quad (6)$$

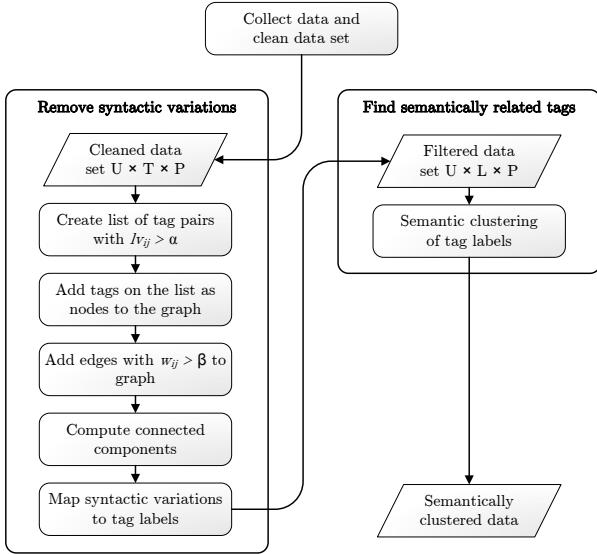


Fig. 1. Overview of the STCS framework

In order to measure the semantic relatedness between tags and the syntactic similarity of short tags, we use the cosine similarity based on co-occurrence vectors. In essence, this measure describes the similarity of the context in which the tags appear. The context here is how often tags are used together with other tags (i.e., the co-occurrence).

3.3 STCS Framework

This section describes the two layers of the STCS framework in detail. An overview of the framework is presented in Fig. 1.

Removing Syntactic Variations In order to remove syntactic variations, we employ an adapted Levenshtein distance measure. The algorithm requires an initial list of tag pairs with a normalized Levenshtein distance above a certain threshold α as input. The α threshold represents the minimum normalized Levenshtein distance for which we consider two tags to be syntactic variations. The initial list is then used to create sets T and E as input for constructing an undirected graph. The set T contains each unique tag on the list. The set E is a set of weighted edges between the nodes in T , where the weight represent the similarities between tags. The weight w_{ij} of an edge in the tag graph is calculated as

$$w_{ij} = z_{ij} \times lv_{ij} + (1 - z_{ij}) \times \cos(\text{vector}(i), \text{vector}(j)), \quad (7)$$

where lv_{ij} is the normalized Levenshtein similarity between tag i and j and

$$z_{ij} = \frac{\max(\text{length}(t_i), \text{length}(t_j))}{\max(\text{length}(t_k))} \in (0, 1], \text{ with } t_i, t_j, t_k \in T. \quad (8)$$

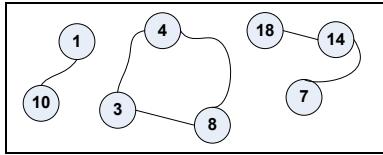


Fig. 2. An example of graph containing three clusters

Using the normalized Levenshtein distances for short tags may result in false positives, i.e., two tags being incorrectly identified as syntactic variations of each other. Therefore, the weight of the cosine similarity between two tags increases as the tags become shorter.

In order to build the tag graph, all the elements from set T are added as nodes to the graph. Subsequently, all the edges in set E for which the weight is above a certain threshold β are added to the graph, connecting the nodes from set T . The β threshold indicates the level of the combined weight measure (w_{ij}) for which we consider two tags to be syntactic variations of each other. Subsequently, the syntactic clusters can be determined by retrieving the connected components in the graph as sets of vertices. A connected component is defined as a maximal subgraph in which all pairs of vertices in the subgraph are reachable from one another. Each subgraph then contains all the nodes (tags) that form a cluster of syntactically related tags. An example of the resulting subgraphs is presented in Fig. 2 with each node containing the id of a unique tag. The resulting graph contains clusters of tags that are syntactic variations of each other. An example of a syntactic cluster is $\{\text{venetie}, \text{venezia}, \text{venzia}, \text{veneza}, \text{venesia}, \text{venizia}\}$, which is assigned the label ‘venice’. We process this data by creating a new data set in which the tags in the clusters are aggregated and presented as a single tag, which we call the label for the cluster. The label of a cluster is the tag which is used most frequently in the data set. The resulting data set is used as input for the semantic clustering layer.

Clustering Semantically Related Tags. After removing the syntactic variations and misspellings from the data set, the new data set can be used to create semantic clusters. For this we use a partitional clustering algorithm, i.e., a clustering-by-committee-based algorithm [19] used by Specia and Motta [21], both with and without some modifications. The choice for this algorithm is motivated by the fact that it uses all tags instead of the cluster centroid to calculate the similarity between two clusters. This allows us to better capture the semantics associated with the tag space. Also, unlike many other clustering algorithms, it allows for multiple classification of tags, which enables us to deal with tag polysemy.

The algorithm starts with creating the initial clusters, where each tag is a separate cluster. Then, all the tags for which the average cosine similarity with respect to all the tags in the clusters is above a certain threshold (χ) are added to the cluster. This could result in many identical or nearly identical clusters. To avoid a high number of these very similar clusters, Specia and Motta use

two smoothing heuristics. For every two clusters, the authors check whether one cluster contains the other, i.e., whether all the elements in the smaller cluster are also present in the larger cluster. If this is the case, the smaller cluster is removed. For each pair of clusters they also evaluate whether the clusters differ within a small margin by checking whether the number of different tags in the smaller cluster with respect to the larger cluster represents less than a percentage of the number of tags in the smaller cluster. If this is the case, the distinct tags in the smaller cluster are added to the larger cluster and the smaller cluster is removed.

A problem with the second heuristic is that the percentage used for merging two similar clusters is constant. This implies that the maximum allowed number of different elements increases with the size of the smaller cluster. Choosing a suitable threshold value is problematic, as we do not want the larger clusters to merge too easily and the smaller clusters too difficultly. The maximum number of different elements for two clusters to be merged, is given by $f(|c|) = \lfloor \epsilon \cdot |c| \rfloor$, where ϵ is the threshold and $|c|$ is the number of elements in the smaller cluster. So for $\epsilon = 0.20$ and $|c| = 30$, the maximum number of different elements is given by $f(30) = 6$. This means that cluster c will be merged into a larger cluster C , if $|D| \leq 6$, where $D = c - C$. Furthermore, as $f(4) = 0$, a cluster with a size below 4 is never merged.

Because of these limitations, we define two new heuristics that replace the original second heuristic. The first new heuristic considers the semantic relatedness of the difference between two clusters. We merge two clusters C and c , where $|C| \geq |c|$, when the average cosine of all elements in D with elements in the larger cluster is above a certain threshold δ . This average cosine is defined as

$$\text{AvgCos} = \sum_{d \in D} \frac{\text{Avg}_d}{|D|}, \text{ with } \text{Avg}_d = \sum_{x \in C} \frac{\cos(x, d)}{|C|}. \quad (9)$$

The second new heuristic considers the size of the difference between two clusters in combination with a dynamic threshold. We merge two clusters in case the normalized difference between the clusters is smaller than a dynamic threshold ϵ . The normalized difference η is defined as

$$\eta = \frac{|D|}{|c|}. \quad (10)$$

Threshold ϵ is defined as

$$\epsilon = \frac{\phi}{\sqrt{|c|}}, \quad (11)$$

and thus

$$f(|c|) = \lfloor \epsilon \cdot |c| \rfloor = \lfloor \phi \cdot \sqrt{|c|} \rfloor. \quad (12)$$

The distribution of the maximum allowed difference for which two clusters are merged can then be adjusted by changing ϕ . An example of a semantic cluster is {london, tatemodernart, towerbridge, millenniumwheel, buckinghampalace, thames}.

As a comparison we also use the algorithm proposed by Lancichinetti et al. [11]. We choose this algorithm because as Specia and Motta's method [21] it allows a tag to be part of multiple clusters. The algorithm uses a graph as an input and attempts to determine the natural community for each node in the graph. In this graph, tags are represented by nodes and weighted edges connect the nodes. The weight of an edge is the cosine similarity of the co-occurrence vectors of the two tags the edge connects. A community is a subgraph G identified by the maximization of the fitness of its nodes. If we consider each tag as a node in the graph, the community of a node forms a cluster of semantically related tags. The fitness of a subgraph G is defined as:

$$f_G = \frac{k_{in}^G}{(k_{in}^G + k_{out}^G)^\theta} \quad (13)$$

where k_{in}^G is the strength of the internal links, which in our case is given by two times the sum of the weights of all edges in G and k_{out}^G is the strength of the external links, which in our case is the sum of the weights of all edges linking nodes in G with nodes not belonging G . The parameter θ is used to adjust the size of the resulting communities. Large values of θ yield very small clusters, while small values result in large clusters. The fitness of node A with respect to graph G is defined as $f_G^A = f_{G+\{A\}} - f_{G-\{A\}}$, where $G+\{A\}$ / $G-\{A\}$ are the graphs obtained from G by adding/deleting node A .

The natural community of a node A is detected as follows. We start with a covered subgraph G including only node A . Each iteration then consists of the following steps:

1. Visit all neighboring nodes of G not included in G ;
2. Add the neighbor with the largest fitness to G , yielding G' ;
3. Recalculate the fitness of each node in G' ;
4. Delete a node that has a negative fitness, yielding G'' ;
5. If a node is deleted in 4, repeat from 3, else repeat from 1 with G being G'' .

This procedure stops when all neighboring nodes considered in step 1 have a negative fitness. However, it is too computationally intensive to perform this procedure for every node. Therefore, the authors describe the following heuristic. First pick a node A at random and detect the community of node A . Next pick a node B not yet assigned to any group and detect the community of this node. This process is repeated until each node is assigned to at least one group.

4 Framework Implementation

This section discusses the implementation of the STCS framework. The implementation of the framework is done in Java in combination with a MySQL database. For data collection and processing we used PHP scripts. This section continues with discussing data collection and processing in Subsection 4.1, and the implementation details for the cosine computation and clustering in Subsection 4.2.

4.1 Data Processing

For our experiments, we gather data from Flickr related to all the pictures uploaded in 2009, together with their associated tags and users. To speed up the data collection process, we distribute this task over four separate machines. The initial data set contains 38,788,518 pictures, 196,344 users, and 1,017,168 tags. After data cleaning, there are 147,064,188 associations, 31,951,884 co-occurrence pairs, and 97,569 tags left.

The data cleaning process consists of several steps that aim to cope with noise encountered in the data due to the lack of restrictions imposed on users assigning tags to pictures. First of all, we remove tags with a tag length larger than 32 characters, to avoid tags that are entire sentences. The number 32 is based on an extensive manual tag analysis. Furthermore, we remove non-Latin characters (e.g., Arabic, Cyrillic, etc.) as well as numeric characters. Subsequently, we remove images from the same user that share identical tag strings. This filter is motivated by the fact that in our data set, we sometimes encounter hundreds of pictures uploaded by the same user with identical tags. These are sets of holiday pictures tagged with identical tags, often unrelated to the picture. To prevent these sets from influencing the co-occurrence measure, we only keep one image of each of these sets and remove the others. Finally, we remove tags which occur in less than 133 different pictures, as they are statistical outliers in our analysis, i.e., $\text{AverageTagOccurrence} - 1.5 \times \text{IQR} \approx 133$.

4.2 Implementation Details

In order to be able to calculate the cosine similarity, one needs the co-occurrence vector for each tag. To obtain this, we construct a matrix with both a row and a column for each tag, with the cells containing the co-occurrence for that particular combination of tags. We aimed to employ the Colt library [5] to store this matrix in memory because of its small memory footprint. However, the size of our matrix is too large to be handled by Colt, and thus we implement our own high performance matrix library which uses a Colt vector to store each column of the co-occurrence matrix. Using the resulting matrix, we calculate the cosine similarity for each unique combination of two tags. Because the co-occurrence matrix is very large and the number of cosine computations increases very fast with the matrix size, we use a distributed system. For this purpose, we utilize Amazon EC2 [1], a service which provides cloud computing resources. We implement the algorithms in a distributed fashion and run them in parallel on multiple high memory instances, each having 17.1 GB of RAM to fit the entire matrix in memory. In our experiments, the total amount of instances running in parallel fluctuate between 3 and 52 instances. In total, these experiments took up 8 computing hours on 2,914,700 cosine similarity calculations for the syntactic clustering, which completed in 2.5 hours of actual time. For the semantic clustering, we use 64 computing hours to perform 50,000,000 cosine similarity calculations within 11 hours of actual time. Each instance loads the full matrix in memory and connects to a central job server which coordinates each instance to perform a distinct portion of the calculations.

In our framework, we implement the syntactic clustering algorithm by means of the Java Universal Network/Graph Framework (JUNG) [18] graph library. This library provides a good method for retrieving the set of connected components in a graph, which are clusters of tags in our case. The semantic clustering algorithms is also written in Java. In order to reduce the time required for the semantic clustering, we only consider the 10,000 most popular tags.

5 Evaluation

This section presents results of experiments conducted on our cleaned Flickr data set. Subsection 5.1 discusses the results related to the first layer of our STCS framework, i.e., removing syntactic variations. Subsection 5.2 elaborates on experimental results related to finding semantically related tags.

5.1 Removing Syntactic Variations

We chose $\alpha = 0.5$ as a threshold for the normalized Levenshtein similarity to identify potential syntactic variations. We chose this constant using a sample of 100 tag pairs that were known to be syntactic variations. We found that the normalized Levenshtein similarity between these tag pairs was never smaller than 0.5. The goal of this threshold was to reduce the number of potential syntactic variations for which the calculation of the cosine similarity was required. A value of 0.5 effectively reduced this number without losing syntactic variations.

For the β threshold we chose a value of 0.7. We have chosen this value because it resulted in the best performance on random samples of 100 clusters. For this threshold we tried all values between 0 and 1 with a step of 0.05. For the evaluation of each value we drew a separate random sample of 100 clusters (our training set). After filtering the cleaned data set discussed in Section 4.1 on syntactic variations, there are 147,064,188 associations, 28,603,077 co-occurrence pairs, and 91,916 tags left.

We evaluate the performance of the clustering technique using the combined measure and the Levenshtein distance using precision and purity. We do not use the density, because in our case it proved to be too computationally intensive. For each clustering technique, we draw a random sample of 100 syntactic clusters and evaluate these manually. For this evaluation we used majority voting with a group of three people. Each person in the group chooses the correct tags in each cluster and majority voting is then used to determine the final number of correct tags that is used in the precision and purity calculations. The average precision of the algorithm used in our framework on the random sample set is 0.89. The average precision of the clustering algorithm using the normalized Levenshtein distance is 0.70. By performing a one-tailed unpaired two sample t-test with a significance level of 0.01, we conclude that the combined measure does perform significantly better than the Levenshtein distance alone in terms of precision per cluster. The purity of the STCS framework is 0.88, while the purity of the technique using only the Levenshtein distance to identify syntactic variations is 0.67. Because we only use one data set and each data set has a

single average precision and purity, it was not possible to perform t-tests for the average precision and purity.

5.2 Finding Semantically Related Tags

We choose $\chi = 0.8$, $\delta = 0.7$, and $\phi = 0.9$ as thresholds for our framework. For the clustering algorithm that uses a constant percentage to identify similar clusters, we set $\epsilon = 0.3$. For the θ threshold used in the method proposed by Lancichinetti et al. [11], we choose $\theta = 1.5$. We used these values because they proved to give the best performance on random samples of 50 clusters (our training set). It is a non-trivial task to evaluate the results of the semantic clustering quantitatively due to the lack of external grounding. Semantic lexicons such as WordNet [8] only contain a small portion of the tags in our data set. We considered all values ranging from 0.1 to 0.9 with an increment of 0.1 for the χ , δ , ϕ and ϵ thresholds and values ranging from 0.5 to 3 with an increment of 0.25 for the θ threshold. We chose this range for θ because smaller values resulted in extremely large clusters and larger values resulted in extremely small clusters. For each threshold we evaluated a separate random sample of 50 clusters for each value. All semantic clustering algorithms utilize the syntactic clustering algorithm using the combined measure to filter out syntactic variations.

For the chosen thresholds we used majority voting and a different random sample of 100 clusters (our test set) for each method to compute the average precision and purity. We perform these computations for the clustering method using the original heuristic for merging two similar clusters, the method using the two new heuristics, and the method introduced by Lancichinetti et al. [11]. For the evaluation we again use majority voting with a group of three people. The average precision when using the two new heuristics is 0.86. The average precision when using the constant percentage as a threshold for merging two clusters is 0.80, and the average precision when using the method introduced by Lancichinetti et al. is 0.81. By performing a one-tailed unpaired two sample t-test with a significance level of 0.05, we conclude that the two new heuristics significantly improve the precision per cluster when compared to the heuristic that uses only the constant percentage. However, with a significance level of 0.05, the method using the new heuristic does not perform significantly better than Lancichinetti's method (w.r.t. precision). The purity of the technique using the two new heuristics is 0.89, while the purity of the technique using the original heuristic for merging two similar clusters is 0.87. The purity of Lancichinetti's method is 0.77. We observe that the purity of the technique with the two new heuristics is the highest, but we cannot test for significance as in our measurements we have purity as a single number (based on only one data set).

6 Conclusion

In this paper, we proposed the STCS framework, which performs syntactic and semantic tag clustering. For the syntactic clustering, we make use of a combined

measure of the Levenshtein distance and the cosine similarity. We compared the results of clustering on the combined measure to clustering on the Levenshtein distance. Our conclusion is that the combined measure performed significantly better in terms on precision. The clustering method as proposed in our framework was able to effectively filter out syntactic variations from the data set.

For semantic clustering, the framework uses an adaptation of the approach proposed by Specia and Motta [21]. We are capable of identifying numerous and useful clusters. Optimizing the parameters is difficult, as it is a non-trivial task to evaluate the results of the semantic clustering quantitatively due to the lack of external grounding, since existing semantic lexicons only contain a small portion of the tags in our data set. Nevertheless, our experiments show that the proposed method significantly outperforms the original method by Specia and Motta and outperforms on average the method of Lancichinetti in terms of precision. Finally, we have shown that our results are valid on a significantly larger data set than was used before in the existing body of literature.

As future work, it could be interesting to investigate how the cluster information can be used to enhance search results and especially how users experience and value this improvement. This would provide crucial insight into which clustering method in the end provides the best results in terms of user experience. We would also like to experiment with the use of the Wikipedia redirects as a tool to help identify syntactic variations of tags. Additionally, the services provided by the TAGora repository [22] might prove useful for identifying syntactic variations.

References

1. Amazon Web Services LLC: Amazon Elastic Compute Cloud, Amazon EC2 (2010), <http://aws.amazon.com/ec2>
2. Arenas, A., Diaz-Guilera, A., Perez-Vicente, C.J.: Synchronization Reveals Topological Scales in Complex Networks. *Phys. Rev. Lett.* 96(11), 1–4 (2006)
3. Begelman, G., Keller, P., Smadja, F.: Automated Tag Clustering: Improving Search and Exploration in the Tag Space. In: Carr, L.A., Roure, D.C.D., Iyengar, A., Goble, C.A., Dahlin, M. (eds.) 15th World Wide Web Conference (WWW 2006), pp. 22–26. ACM Press, New York (2006)
4. Cattuto, C., Benz, D., Hotho, A., Stumme, G.: Semantic Grounding of Tag Relatedness in Social Bookmarking Systems. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayanan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 615–631. Springer, Heidelberg (2008)
5. CERN - European Organization for Nuclear Research: Colt Libraries for High Performance Scientific and Technical Computing in Java (2010), <http://acs.lbl.gov/~hoschek/colt/>
6. Delling, D., Gaertler, M., Görke, R., Nikoloski, Z., Wagner, D.: How to Evaluate Clustering Techniques. Tech. rep., Faculty of Informatics, Universitat Karlsruhe (2006), <http://digbib.ubka.uni-karlsruhe.de/volltexte/1000007104>
7. Echarte, F., Astrain, J.J., Córdoba, A., Villadangos, J.: Pattern Matching Techniques to Identify Syntactic Variations of Tags in Folksonomies. In: Lytras, M.D., Carroll, J.M., Damiani, E., Tennyson, R.D. (eds.) WSKS 2008. LNCS (LNAI), vol. 5288, pp. 557–564. Springer, Heidelberg (2008)

8. Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
9. Golder, S., Huberman, B.: The Structure of Collaborative Tagging Systems. Tech. rep., Information Dynamics Lab, HP Labs (2005),
<http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0508082>
10. Jäschke, R., Hotho, A., Schmitz, C., Stumme, G.: Analysis of the Publication Sharing Behaviour in BibSonomy. In: Priss, U., Polovina, S., Hill, R. (eds.) ICCS 2007. LNCS (LNAI), vol. 4604, pp. 283–295. Springer, Heidelberg (2007)
11. Lancichinetti, A., Fortunato, S., Kertesz, J.: Detecting the Overlapping and Hierarchical Community Structure in Complex Networks. *New Journal of Physics* 11(3), 1–19 (2009)
12. Larsen, B., Aone, C.: Fast and Effective Text Mining using Linear-Time Document Clustering. In: 5th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 1999), pp. 16–22. ACM, New York (1999)
13. Levenshtein, V.I.: Binary Codes Capable of Correction Deletions, Insertions, and Reversals. *Soviet Physics Doklady* 10(8), 707–710 (1966)
14. Manning, C., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
15. Markines, B., Cattuto, C., Menczer, F., Benz, D., Hotho, A., Stumme, G.: Evaluating Similarity Measures for Emergent Semantics of Social Tagging. In: 18th World Wide Web Conference (WWW 2009), pp. 641–650. ACM, New York (2009)
16. Mika, P.: Ontologies Are Us: A unified model of social networks and semantics. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 522–536. Springer, Heidelberg (2005)
17. Newman, M.E.J., Girvan, M.: Finding and Evaluating Community Structure in Networks. *Physical Review E* 69(2), 1–15 (2004)
18. O'Madadhain, J., Fisher, D., Nelson, T., White, S., Boey, Y.B.: Java Universal Network Graph (JUNG) Framework (2010), <http://jung.sourceforge.net>
19. Pantel, P.: Clustering by Committee. Ph.D. thesis, University of Alberta (2003)
20. Schachter, J.: Delicious - Social Bookmarking (2010), <http://www.delicious.com/>
21. Specia, L., Motta, E.: Integrating Folksonomies with the Semantic Web. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 624–639. Springer, Heidelberg (2007)
22. TAGora: TAGora Sense Repository (2010),
<http://tagora.ecs.soton.ac.uk/tsr/index.html>
23. van Dam, J.W., Vandic, D., Hogenboom, F., Frasincar, F.: Searching and Browsing Tag Spaces Using the Semantic Tag Clustering Search Framework. In: Fourth IEEE International Conference on Semantic Computing (ICSC 2010), pp. 436–439. IEEE Computer Society, Los Alamitos (2010)
24. Yeung, C., Gibbins, N., Shadbolt, N.: Contextualising Tags in Collaborative Tagging Systems. In: 20th ACM Conference on Hypertext and Hypermedia (HT 2009), pp. 251–260. ACM, New York (2009)

A Semantic Web Annotation Tool for a Web-Based Audio Sequencer

Luca Restagno¹, Vincent Akkermans², Giuseppe Rizzo¹, and Antonio Servetti¹

¹ Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy,
`luca.restagno@studenti.polito.it`,

`{giuseppe.rizzo,antonio.servetti}@polito.it`,

² Music Technology Group, Universitat Pompeu Fabra, Barcelona, Spain,
`vincent.akkermans@upf.edu`

Abstract. Music and sound have a rich semantic structure which is so clear to the composer and the listener, but that remains mostly hidden to computing machinery. Nevertheless, in recent years, the introduction of software tools for music production have enabled new opportunities for migrating this knowledge from humans to machines. A new generation of these tools may exploit sound samples and semantic information coupling for the creation not only of a musical, but also of a “semantic” composition. In this paper we describe an ontology driven content annotation framework for a web-based audio editing tool. In a supervised approach, during the editing process, the graphical web interface allows the user to annotate any part of the composition with concepts from publicly available ontologies. As a test case, we developed a collaborative web-based audio sequencer that provides users with the functionality to remix the audio samples from the Freesound website and subsequently annotate them. The annotation tool can load any ontology and thus gives users the opportunity to augment the work with annotations on the structure of the composition, the musical materials, and the creator’s reasoning and intentions. We believe this approach will provide several novel ways to make not only the final audio product, but also the creative process, first class citizens of the Semantic Web.

Keywords: Ontology driven annotation tool, Semantic audio annotation, web-based audio sequencer, Semantic Web sequencer, Semantic Web.

1 Introduction

One of the key focus of the Semantic Web [1] is the process of combining data from different sources in order to easily elaborate that information programmatically. The aggregation of data extends the available information about a resource. In this context, the process of tagging or annotation, performed by human beings, plays an important role in achieving a more precise definition of the content. Even though the practice of tagging content has become widespread

among many Internet user communities, it is affected by several problems including the inconsistency of terms used to annotate contents. Then, one of the challenges in this research field is to ensure consistency across annotations in the choice of terms and their meaning. A solution is to restrict the user's annotations to a certain set of concepts defined in an ontology. Generally, an ontology defines a domain description that can be used as a controlled vocabulary and that may help in the process of context enrichment and crowdsourced content classification. On the other side, an obvious disadvantage of this approach is its possible incompleteness due to missing concepts, a problem that does not exist in the free tagging approach. However, by combining several well designed ontologies users may be guided to create high quality annotations.

The ontology driven annotation framework detailed in this paper is developed to allow users to annotate sonic material in a controlled way, using concepts extracted from different ontologies. The main feature of the framework is the capability to load from the Web any ontology specified in the Web Ontology Language (OWL) [11] and to guide users through the process of content information enrichment. When an ontology is loaded, the annotation framework allows the creation of annotations that link selected parts of the content to the concepts of that ontology. In addition, it creates a RDF graph for each annotation which is published so that annotation is available to everyone. Furthermore, the annotation framework has been completely developed using Web standard languages, in such a way that it can be easily used inside other web applications. Moreover, it provides a user-friendly web front-end to make the semantic annotation an easy task. In order to accomplish this goal, the framework loads all classes and their attributes from the ontology; then classes are easily browsable through the widgets of the Web user interface. For each attribute of a class, the tool loads the right user interface widget to let the user specify a value. Furthermore, the annotation framework has been integrated in a web-based audio sequencer in order to allow annotation of audio contents, realizing a complete environment for composing and annotating sounds. Through the Web interface, users may play sequentially or may remix sounds available on the Web. The sound files are loaded into the sequencer simply by specifying their URLs. For example our audio material has been retrieved from Freesound¹, a collaborative database of Creative Commons licensed sounds. Then, loading a specific ontology, users are guided in the process of annotating the gathered sounds, enriching the description of the composition created and, finally, the meaning.

Music and sound have a rich semantic structure. They communicate a message, designed by the composer or sound maker, which ranges from nonsensical or abstract to symbolic (e.g. a piece of film music supporting a clear narrative). A majority of media production nowadays is done with software tools, which give rise to various new opportunities to monitor the production process. In this work we focused on the combination of the annotation framework and the web audio sequencer, to investigate the implications of this idea. Take for example video games, which generally have a non linear narrative that is often supported

¹ <http://www.freesound.org>

by affective music. As the game world can give rise to a variety of situations the music should be able to adapt. If the composer is able to formally describe pieces of his non linear composition by using our annotation framework and the right ontology, the game would be able to generate new music by matching the formalized intentions in both domains. Another example is music education, where providing insight and understanding into the music is the primary concern. A piece of music, whose structure and different aspects have been formally annotated, could be represented in different ways and thus give students views on the work of art that match their capabilities or interests. Possibly with future work the artist can, while working, develop his own ontology. This ontology would describe the themes of the work as seen by the creator and allow himself and others to reflect and learn from this.

The remainder of this paper is organized as follows: a review of the current state of the art is presented in section 2, key ideas of our approach are introduced in section 3 and the description of the annotation tool is showed in section 4. A contextualization of the tool, by means of a use case, is described in section 5, followed by conclusions and future work in section 6.

2 Related Work

The process of media content enrichment by means of user generated data is a challenging task that has been addressed by the Semantic Web community in several works. An often used approach is to give users the ability to annotate a content in order to more easily retrieve the information later. In particular, annotation is often required to refine and improve data descriptions when automatic feature extraction tools are employed. In this context, many efforts have been done to generate annotations in order to point to and, then, retrieve contents. Our work takes a different approach and combines the annotation tool with a web sequencer in order to provide the ability to remix sound files and to enrich the newly created composition with a controlled vocabulary. In the rest of this section we give an overview of the state of the art in the media content annotation tools.

In the LabelMe Project [14], Russell et al. produced a web-based image annotation tool to identify and define specific objects inside images. Their goal was to provide a dynamic dataset for object recognition and computer graphics. Although this research worked with tags, they did not focus on the annotation itself. Additionally, they used free textual tags, so annotations were affected by the typical folksonomy [4] problems, like polysemy and synonymy. In our approach we avoid these problems by using a controlled vocabulary, an ontology chosen by the user during the startup process, which includes a concept hierarchy from a specific knowledge domain.

The use of controlled vocabulary to improve semantic annotation of images has been explored in the M-OntoMat-Annotizer [12]. Petridis et al. implemented different ontologies based on the MPEG-7 standard to let users associate visual descriptors with content. This work permits selection of a part of an image or

frame and association of a concept retrieved from the provided ontology. Based on this work, we take advantage of multiple formalized ontology domains to extend the descriptive possibilities of an annotator.

In [16], Wang et al. addressed the annotation problem by means of a set of ontologies which are linked using a bridge ontology. This work overcame the problem of ontology reuse and prevented unnecessary ontology extension and integration. Although this idea is promising, it presents only a general idea of how to link ontologies without proposing a clear method. However, G. Kobilarov et al. [8], following [16], used a bridge ontology to categorize and link to DBpedia [2] multimedia documents located within the BBC archives. By means of this approach, they exploited object persistence in the BBC categorization system (CIS) and mapped resources according to DBpedia references: resource disambiguation is achieved and semantic information is augmented. Similarly to this approach, we provide the possibility to take advantage of multiple ontologies, but we do not try to link concepts between them. Our tool permits the use of concepts from a single online available ontology at a time. The ontology can, however, be switched whenever the annotator feels the need.

In order to facilitate annotation and sharing of annotations among many users, we implemented a web-based solution. An attempt to distribute annotations over the Web is represented by the Annotea Project [6], which aims to provide a system to share annotations on a general-purpose open RDF infrastructure. It suggests a possible set of technologies to implement a semantic web infrastructure for creating, editing, viewing and sharing annotations. In [15], Ronald Schroeter et al., proposed how to use the Annotea Schema to extend annotation links among multiple video resources. They used annotation to highlight parts of video and mapped them in the MPEG-7 standard. Pointing to different highlighted parts of video may make a new composition, which could be built automatically and could be reproduced by users: this is the idea behind the work of Rene Kaieser et al [7]. Although this problem is relevant for the research community, our work differs from it because here the user supervises the enrichment process. A similar use case was addressed by Maria Meleshkova et al. [9] where Web APIs are annotated by users with an annotation editor. Finally, we use some of the framework technologies, like the Annotea Annotation RDF Schema, and we developed a web-based framework for the ontology-driven annotation of audio contents.

3 Rationale

Our semantic web annotation tool is a web tool for annotating any kind of resource. It can load any online available OWL ontology and guide users through the annotation process with a simple user interface. When starting the annotation process the user is allowed to select the ontology that deals with the aspect of the resource he wants to make statements about. Additionally we assess our tool with annotations of sonic material. In this context we present an open web framework that can improve the user experience during the creative process, by means of ontology driven annotation.

Communities, that use the free text annotation method, are affected by a set of problems, like polysemy, synonymy, data scarcity, spelling errors and plurals. Polysemous tags can return undesirable results. For example, in a music collection when a user searches for the tag *love*, results could contain both love songs and songs that were tagged as such because user liked them very much. Tag synonymy is also an interesting problem. Even though it enriches the vocabulary, it also presents inconsistencies among terms used in the annotation process. According to [10], *bass drum* sounds can be annotated with the *kick drum* tag, but these sounds will not be returned when searching for *bass drum*. To avoid this problem, sometimes users tend to add redundant tags to facilitate the retrieval (e.g. using *synth*, *synthesis* and *synthetic* for a given sound).

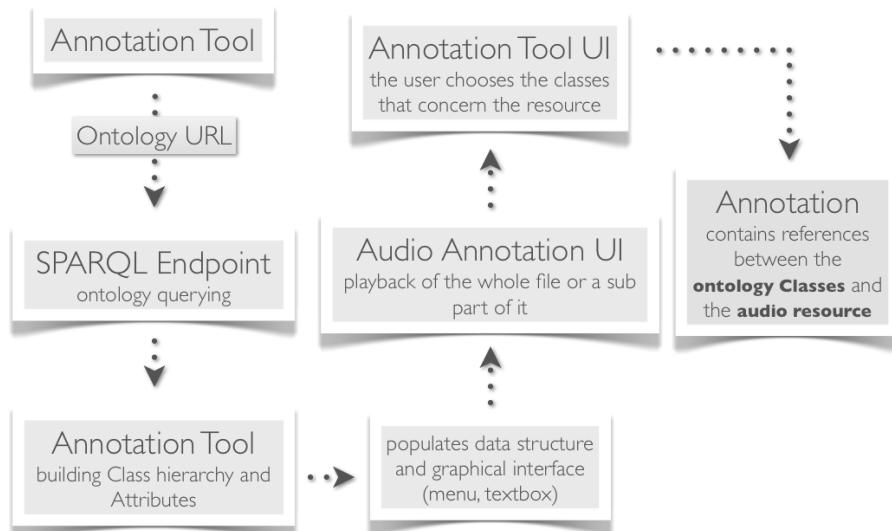


Fig. 1. The annotation tool retrieves a set of concepts from knowledge repositories (ontologies/taxonomies). Then it exposes the set of concepts to the annotator through the graphical interface. Using this front-end, the user can link semantic concepts to an audio resource whose contents were previously unknown. The result is an annotation, a document that stores the links the human annotator creates.

Figure 1 shows how our tool works. It retrieves information from knowledge repositories available on the Web. They can be formalized as ontologies, like the large and popular Music Ontology [13], an attempt to link all the information about musicians, albums and tracks together. It can exploit ontologies specifically developed for an application. Moreover, it can use Web databases that provides a query service based on the semantic web technologies, like the DBpedia project that allows access to the large database of Wikipedia via semantic web resources. Pursuing this approach we wanted to conform to the Linked Data principle [5] of distributable and connected pieces of information. The user can

associate with the resource any concept of the ontology, in order to extend the semantic description of the digital content.

The tool provides an intuitive Web user interface that lets users choose one of the classes in the ontology. When the user is done annotating, the annotations are converted to the RDF syntax. As example, in Figure 3 is shown their serialization by means of RDF/XML syntax. These are then sent to the server and saved in a triple store, ready to be retrieved and queried.

```

<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:an="http://www.w3.org/2000/10/annotation-ns#"
  xmlns:ext-owl="http://mtg.upf.edu/2010/02/SoundProducingEvents.owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <an:Annotation>
    <an:annotates>
      <rdf:Description>
        <an:source rdf:resource="http://example.org/audio.mp3"/>
      </rdf:Description>
    </an:annotates>
    <an:body>
      <ext-owl:Vibrating_objects></ext-owl:Vibrating_objects>
    </an:body>
    <an:created>2010-02-22T00:04:26Z</an:created>
  </an:Annotation>
  <an:Annotation>
    <an:annotates>
      <rdf:Description>
        <an:source rdf:resource="http://example.org/audio2.mp3"/>
      </rdf:Description>
    </an:annotates>
    <an:body>
      <ext-owl:Impact>
        <ext-owl:configuration rdf:parseType="Literal"> -1 </ext-owl:configuration>
        <ext-owl:material rdf:parseType="Literal"> 2 </ext-owl:material>
        <ext-owl:size rdf:parseType="Literal"> 3 </ext-owl:size>
        <ext-owl:mallet_hardness rdf:parseType="Literal"> 4 </ext-owl:mallet_hardness>
        <ext-owl:force rdf:parseType="Literal"> 1.01034 </ext-owl:force>
      </ext-owl:Impact>
    </an:body>
    <an:created>2010-02-22T00:04:26Z</an:created>
  </an:Annotation>
</rdf:RDF>
```

Example 3 - RDF/XML annotations generated by our annotation tool.

4 Annotation Tool

The annotation tool consists of a client side and a server side component.

4.1 Client-Side Component

The client-side component is a graphical user interface consisting of boxes, menus and input fields to let the user navigate the classes provided by the ontology.

It also allows to choose one or more classes, and specifying the value for the attributes of a class, if present. According to the SoC (separation of concerns) guidelines, we developed our tool using HTML for page markup, CSS for graphical style and JavaScript to handle the program logic and user interactions. The jQuery framework² was used to manipulate the Document Object Model (DOM) and the jQuery UI³ utilized for the GUI components like autocomplete, datepickers and complex behaviour handlers like draggable and droppable. The tool has been developed with attention to modular programming. In order to allow other developers to reuse the code, our annotation tool was divided into three reusable modules: owl.js, owl-ui.js and owl-ui.audio.js.

- owl.js: requests an interpretation of a specified ontology from the server side component and converts this to an internal data model.
- owl-ui.js: is responsible for the creation of the annotation tool panel, composed of menus and dynamic textboxes. It requires the owl.js library to populate the user interface widgets with the information retrieved from the ontology.
- owl-ui.audio.js: creates an interface to annotate audio files. It allows the user to listen to files and, using the audio waveform image, to select a sub part of a sound in order to annotate it. Then, it allows opening of the annotation tool panel generated by the owl-ui.js library in order to annotate a sound with ontology classes.

These libraries can be embedded into any web page, making it particularly easy for a developer to add the annotation feature to his own web application. Furthermore, it would be relatively easy to develop special user interface components for annotating other types of documents, like video or text.

4.2 Server-Side Component

The second part of our tool is the server-side component. It is a SPARQL Protocol and RDF Query Language (SPARQL) endpoint which makes queries over the ontology and retrieves all classes, properties and attributes. The response is generated in JavaScript Object Notation (JSON) format (but it is possible to request different output formats, like raw text and XML) and it is returned to the client side. Example 4.2 provides a sample response. JSON notation is used because it is a lightweight data-interchange format, it is readable by humans and can be easily converted from text to JavaScript object. The SPARQL endpoint runs on a Linux machine with the Apache 2 web server running and the PHP language available. Furthermore, the endpoint uses the Redland RDF libraries to interpret the data from the ontologies and they are a key component of our framework.

² <http://jquery.com/>

³ <http://jqueryui.com/>

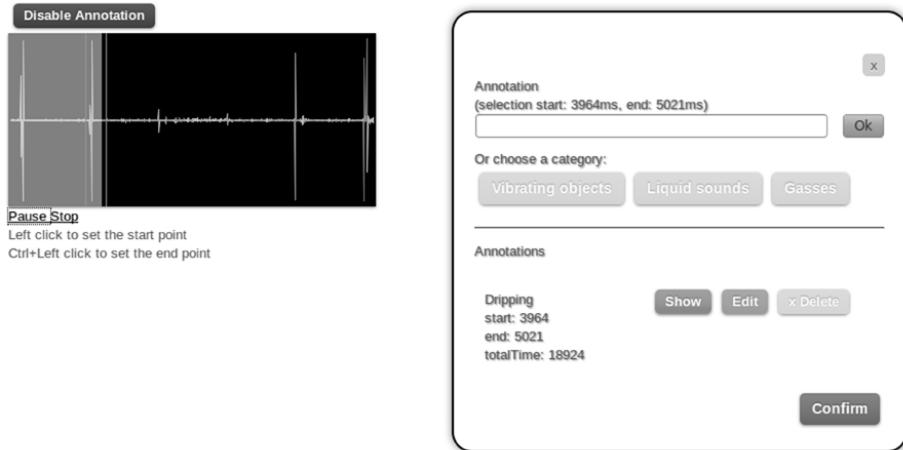


Fig. 2. An example of audio annotation using the annotation tool. On the left the waveform of a sound retrieved from Freesound, reproducing a dripping faucet. On the right the annotation tool front-end where the user is linking the Dripping category of Liquid sounds, to a temporal interval of the sound. The categories are provided by the Sound Producing Events Ontology that is loaded from the Web.

```
{
  "head": {
    "vars": [ "parentClass", "subClass" ]
  },
  "results": {
    "ordered" : false,
    "distinct" : false,
    "bindings" : [
      {
        "parentClass" : { "type": "uri", "value": "http://www.example.org/ChordTaxonomy.owl#Triad" },
        "subClass" : { "type": "uri", "value": "http://www.example.org/ChordTaxonomy.owl#Minor" }
      },
      ...
    ]
  }
}
```

Example 4.2 - JSON serialization of a SPARQL query result. This query returns each class of the ontology and a sub class of it.

4.3 Annotation Process Details

Figure 1 shows the annotation tool flow chart. When our annotation tool is initialized, it makes a synchronous call to the SPARQL Endpoint hosted by a server machine and it sends three main parameters: the URL of the ontology to query, the SPARQL query to execute and the format of the response.

The annotation tool receives a response, by default a JSON object, containing each class and subclass of the ontology. Processing this data our tool creates a

JavaScript structure of objects containing the complete ontology hierarchy of classes. At this point the annotation tool creates the user interface populated with data retrieved from the ontology. The resulting GUI widget includes a textbox, in which dynamic suggestions are provided by means of the autocomplete feature. The user can traverse the class hierarchy through a tree menu to choose a concept related to the resource he is annotating. When the user selects a class from the menu or from the textbox he is presented with a new widget where the user can assign a value to each attribute of the class. The tool chooses the right widget for each possible attribute type.

The annotations are collected into a stack and when the annotation process is completed, the user confirms the annotation. The tool generates an RDF representation of the annotations and subsequently sends it to a server where it could be stored in a triple store and retrieved later. Thanks to namespaces and URIs that identify uniquely a resource, the generated RDF/XML annotation holds the complete semantic description and the information the user has associated with the resource.

4.4 An Example of Audio Annotation

In this section we illustrate the widgets developed to allow users to annotate audio resources. In this specific case, we retrieve a dripping faucet sound and its waveform image from the Freesound repository.

As shown in Figure 2, a user may playback the entire sound and select a part of it by clicking on the audio's waveform. This way the annotator can link multiple different concepts to the same sound, or even to particular events that occur within the sound recording. In this case we loaded the Sound Producing Events Ontology, based on the work of W. W. Gaver [3]. In his book “What in the world do we hear? an ecological approach to auditory event perception” he proposed a framework for describing sound in terms of audible source attributes. We formalized a possible ontology based on the work of Gaver, using the Web Ontology Language (OWL) and published it on the Web in the form of an RDF graph. The ontology used can be easily substituted by the developer, specifying the URL of another ontology, so that classes coming from the new repository can be available to the widgets, ready to be linked to a digital resource.

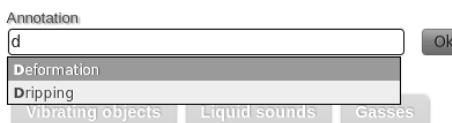


Fig. 3. The annotation tool offers intuitive widgets to find the right ontology concepts. The image illustrates a text field with an autocomplete feature.

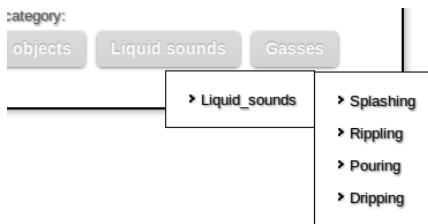


Fig. 4. The annotation tool allows navigation through the class hierarchy with a dynamic interactive menu

The annotation tool front-end is composed of a text field with autocompletion (as shown in Figure 3), so that by typing a user receives suggestions on the available concepts. Alternatively, a user can traverse the complete concept hierarchy through a tree menu (as shown in Figure 4) which illustrates the concept relationships and that appears by clicking on the root concepts (in this case Vibrating objects, Liquid sounds and Gasses).

On the lower part of the user interface there is a stack of concepts the user already linked to the resource. It allows editing of attributes of the concept, showing annotated sound parts and in addition permits to delete an annotation previously created. Clicking on the Confirm button, the annotation tool generates an RDF/XML GRAPH which stores all the links and the information between the OWL classes and resources. This graph can be easily stored and retrieved later.

5 Use Case: A Web-Based Audio Sequencer

In order to test the capabilities of our annotation tool in terms of usability and reliability, we developed a web-based audio sequencer for the Web, where users can work with sounds, mixing and annotating them in a production environment. The tool is available at the test project web page⁴. This site implements the annotation tool technology described above and it works as a hub, because it refers to sound files hosted by the Freesound website.

We chose to realize a web-based audio sequencer completely developed with standard web languages. We used the standard mark-up language designed for the Web, HTML, graphical customization allowed by CSS stylesheets and we handled the business logic and user interactions with JavaScript. We also tried to exploit the multimedia capabilities of the new version of the HTML standard, but our project required advanced audio synchronization features that HTML5 Audio does not yet provide. We had to fall back to Adobe Flash technology that is responsible for handling audio playback. Figure 6 shows three different layers of our application:

⁴ <http://pittore.polito.it:8080/wcs/>

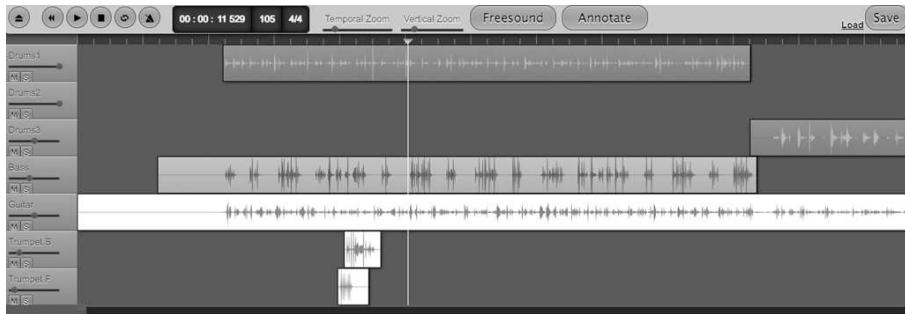


Fig. 5. The web sequencer user interface is similar to a professional audio editing program. It permits synchronization of tracks by dragging them on the grid, controlling the audio playback of the composition and zooming the view. It also implements searching of the Freesound large database of sounds. Thanks to the integration of the annotation tool, it is possible to describe each sound event of the composition with accuracy.

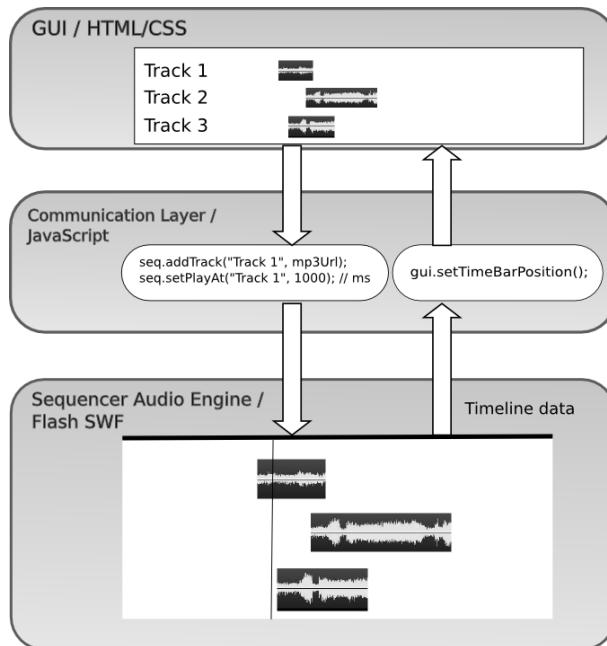


Fig. 6. The architecture of the web sequencer is composed of three layers: the Audio Engine, the Communication Layer and the Graphical Interface

- The Audio Engine Layer is responsible for the playback of the audio. It retrieves audio files from the Web, then it synchronizes tracks and handles the virtual timeline. In addition, it has features to mute and solo a track and change its volume. It also permits looping a section of the composition

and includes a metronome. It communicates with the Communication Layer described below.

- The Communication layer controls bidirectionally the Audio Engine Layer and the Graphical Interface of the application. When a user performs an action, it is handled by the Communication Layer that transmits the instructions to the Audio Engine Layer. It also receives events from the Audio Engine Layer and updates the User Interface.
- The Graphical User Interface handles all interactions with the user through drag functionalities, buttons, sliders and editable text fields. When a user interacts with the GUI, the Communication Layer propagates the action to the Audio Engine.

Through this application users can mix sounds available over the web simply using the URL of the audio resources. It implements the basic functionalities of every sequencer, like audio playback, visual tracks synchronization and looping, so that users can create their own audio composition. We also integrated searching of the Creative Commons licensed sound repository Freesound, so that users can retrieve sounds from a large repository.

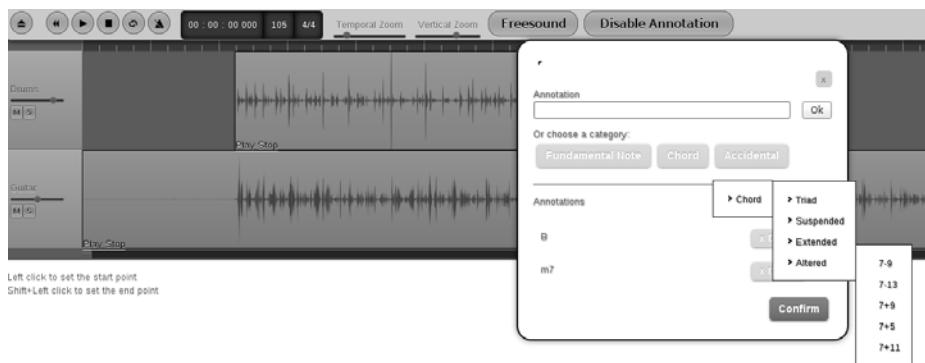


Fig. 7. Integration between the Web audio sequencer and the annotation tool. In this case the user loaded the Chord Taxonomy, in order to describe the harmony of the musical samples included in the composition.

As shown in Figure 7 the annotation tool has been integrated into this sequencer. A user can use it to give the semantic description of an audio file content. By clicking on the Annotation button the user has the opportunity to choose an argument: every argument corresponds to an existing ontology. The current implementation proposes the Sound Producing Events ontology recommended for natural sounds and the taxonomy on Chords, useful to describe the harmony of a music track. Adding new ontologies is really simple for a developer, due to the fact that every ontology is an OWL file located over the Web. After this choice, the GUI is enriched by some new buttons and text. At this point the user can select a portion of an audio sample from the sequencer composition and

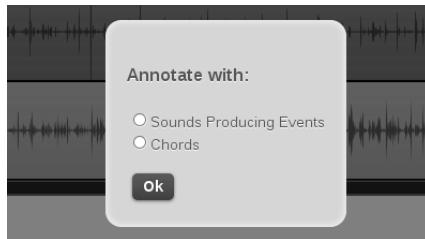


Fig. 8. In our use case, we propose two ontologies. The first is useful in annotating natural sounds, while the second is useful in describing the harmony aspect of a music track.

choose to annotate it. Then, from the annotation tool panel that appears on the screen, he or she can listen to portions of the samples and select concepts that better describe the content (Figure 8). In this way we improve the granularity of the annotation, enriching the semantic description of an audio resource.

The technologies used permit easy integration of the annotation tool on every website. What is needed is to include the Javascript libraries into the site code. Furthermore, the possibility to plug in any ontology available on the web makes the tool a possibly useful instrument for web sites that want to include annotation functionality.

6 Conclusions and Future Work

In recent years, the introduction of software tools for music production have enabled new opportunities for migrating this knowledge from humans to machines. A new generation of these tools may exploit sound samples and semantic information aggregation for the creation not only of a musical, but also of a “semantic” composition. In this paper we have presented a semantic annotation tool integrated with a web-based audio sequencer. A novel approach is used to manipulate multiple audio contents in a dynamic way through a web front-end by simply pointing to their semantic references. The value of the annotation tool lies in the fact that it can load any OWL ontology and guide the end user in the annotation process. Using an easy-to-use graphical interface, it permits the annotator to make links among formalized concepts and resources not described yet, in order to increase the global knowledge of the contents on the web. Furthermore, we realized a collaborative web-based audio sequencer to test our tool. With it, users can remix sounds from the Freesound website and annotate them. The project has followed web standards and principles with the goal of making music compositions part of the Semantic Web.

Future plans consist in extending the tool by developing new user interfaces to annotate other types of resources. We focused on the annotation of sound and music, but it is possible to easily implement visual interfaces for the annotation of video, images and text documents. The tool has been developed to be modular, so that it is not necessary to modify the core libraries. Furthermore we would like to

improve the implementation of the web front-end in order to make it completely cross-browser and cross-platform. An evaluation of this tool in a user community may help to draw user trends, performing a classification of interactions. In addition, some work will be devoted to integrate an automatic tagging process, by means of a text mining algorithm with the annotation supervised by a human.

Acknowledgment

This project was conducted at the Music Technology Group of the Universitat Pompeu Fabra in Barcelona, and in collaboration with the Dipartimento di Automatica e Informatica of the Politecnico di Torino.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. *Scientific American*, 34–43 (May 2001)
2. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: Dbpedia - a crystallization point for the web of data. *Web Semant.* 7, 154–165 (2009)
3. Gaver, W.W.: What in the world do we hear? an ecological approach to auditory event perception. *Ecological Psychology* 5, 1–29 (1993)
4. Halpin, H., Robu, V., Shepherd, H.: The complex dynamics of collaborative tagging. In: Proceedings of the 16th International Conference on World Wide Web, WWW 2007, pp. 211–220. ACM, New York (2007)
5. Hausenblas, M.: Exploiting linked data to build web applications. *IEEE Internet Computing* 13, 68–73 (2009)
6. Kahan, J., Koivunen, M.R.: Annotea: an open rdf infrastructure for shared web annotations. In: Proceedings of the 10th International Conference on World Wide Web, WWW 2001, pp. 623–632. ACM, New York (2001)
7. Kaiser, R., Hausenblas, M., Umgeher, M.: Metadata-driven interactive web video assembly. *Multimedia Tools and Applications* 41, 437–467 (2009)
8. Kobilarov, G., Scott, T., Raimond, Y., Oliver, S., Sizemore, C., Smethurst, M., Bizer, C., Lee, R.: Media meets semantic web – how the BBC uses dBpedia and linked data to make connections. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 723–737. Springer, Heidelberg (2009)
9. Maleshkova, M., Pedrinaci, C., Domingue, J.: Semantic annotation of web apis with sweet. In: 6th Workshop on Scripting and Development for the Semantic Web, Colocated with ESWC 2010 (2010)
10. Martnez, E., Celma, O., Sordo, M., De Jong, B., Serra, X.: Extending the folksonomies of freesound.org using content-based audio analysis. In: Sound and Music Computing Conference, Porto, Portugal (July 23, 2009)
11. Motik, B., Patel-Schneider, P.F., Parsia, B.: Owl 2 web ontology language: Structural specification and functional-style syntax (2009), <http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/>
12. Petridis, K., Anastopoulos, D., Saathoff, C., Timmermann, N., Kompatsiaris, Y., Staab, S.: M-ontoMat-annotizer: Image annotation linking ontologies and multi-media low-level features. In: Gabrys, B., Howlett, R.J., Jain, L.C. (eds.) KES 2006. LNCS (LNAI), vol. 4253, pp. 633–640. Springer, Heidelberg (2006)

13. Raimond, Y., Sutton, C., Sandler, M.: Interlinking music-related data on the web. *IEEE Multimedia* 16, 52–63 (2009)
14. Russell, B., Torralba, A., Murphy, K., Freeman, W.: Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision* 77, 157–173 (2008)
15. Schroeter, R., Hunter, J., Newman, A.: Annotating relationships between multiple mixed-media digital objects by extending annotaea. In: Francconi, E., Kifer, M., May, W. (eds.) *ESWC 2007. LNCS*, vol. 4519, pp. 533–548. Springer, Heidelberg (2007)
16. Wang, P., Xu, B.W., Lu, J.J., Kang, D.Z., Li, Y.H.: A novel approach to semantic annotation based on multi-ontologies. In: *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, 2004, vol. 3, pp. 1452–1457 (August 2004)

CloudFuice: A Flexible Cloud-Based Data Integration System

Andreas Thor¹ and Erhard Rahm²

¹ University of Maryland Institute for Advanced Computer Studies, USA

² University of Leipzig, Department of Computer Science, Germany

`thor@umiacs.umd.edu, rahm@informatik.uni-leipzig.de`

Abstract. The advent of cloud computing technologies shows great promise for web engineering and facilitates the development of flexible, distributed, and scalable web applications. Data integration can notably benefit from cloud computing because integrating web data is usually an expensive task. This paper introduces CloudFuice, a data integration system that follows a mashup-like specification of advanced dataflows for data integration. CloudFuice’s task-based execution approach allows for an efficient, asynchronous, and parallel execution of dataflows in the cloud and utilizes recent cloud-based web engineering instruments. We demonstrate and evaluate CloudFuice’s applicability for mashup-based data integration in the cloud with the help of a first prototype implementation.

Keywords: Cloud Data Management, Data Integration, Mashups.

1 Introduction

Cloud computing technologies shows great promise for web engineering. Distributed data stores (e.g., Google’s Bigtable), web-based queue services (e.g., Amazon SQS), and the ability to employ computing capacity on demand (e.g., Amazon EC2) facilitate the development of flexible, distributed, and scalable web applications. Furthermore, recent efforts in entity search engines [5] and the cloud of linked data [2] have lowered the barriers to easily access huge amounts of data.

Data integration [16] can notably benefit from cloud computing because accessing multiple data sources and integration of instance data are usually expensive tasks. For example, entity matching [14], i.e., identification of entities referring to the same real-world object, is key for linking multiple data sources. Since web data is usually dirty, sophisticated matching techniques must employ multiple similarity measures to make effective match decisions. The pair-wise similarity computation usually has quadratic complexity and is thus very expensive for large-scale data integration. On the other hand, data of interest is usually obtained by sending queries to external data sources. However, the use of existing search engines may require several queries for more complex integration tasks to obtain a sufficient number of relevant result entities [9]. Execution of

many queries needs to be reliable, i.e., it requires handling of failed queries (e.g., due to network congestion) as well as dealing with source restrictions, such as access quota.

Mashup-based data integration [17] demonstrates a programmatic and data-flow-like integration approach which is complementary to common query- and search-based data integration approaches, e.g., data warehouses or query mediators. In fact, mashups are built on the idea of combining existing services so that they can also use existing search engines and query services. However, current mashup dataflows are mostly comparatively simple and do not yet exploit the full potential of programmatic data integration, e.g., as needed for enterprise applications or to analyze larger sets of web data. Although cloud services make it easy to host a mashup application on multiple servers in the cloud, the development of mashup dataflows that can transparently run on multiple nodes still requires substantial development effort.

In this paper we present CloudFuice, a data integration system that allows for the specification and execution of advanced dataflows for data integration that can be employed within mashups. It utilizes a simple access model for external sources to easily incorporate common web sources such as entity search engines, HTML pages, or RDF data sources. A script language provides operators for common data integration tasks such as query generation and entity matching. Beside fast script development the CloudFuice approach also strives for an efficient execution of dataflows in the cloud. Many integration scenarios may easily involve thousands of entities that need to be processed and thus demand scalability. CloudFuice supports an asynchronous and parallel execution of scripts on multiple machines in a cloud as well as elastic computing, i.e., available cloud capacities can immediately be used when they become available.

In this paper, we make the following contributions:

- We introduce a powerful data integration script language that is tailored for web data integration dataflows. (Section 2)
- We detail how a dataflow can be transformed into independent tasks that in turn can be executed asynchronously and in parallel in the cloud. (Section 3)
- We present the CloudFuice architecture facilitating dataflow execution in the cloud. We describe our prototype that also serves as a mashup development tool. (Section 4)
- We evaluate CloudFuice’s parallel and asynchronous execution approach and demonstrate that it can significantly reduce the execution time of dataflows. (Section 5)

We discuss related work in Section 6 before we conclude.

2 Dataflow Definition

In this section we describe how a developer can specify data integration dataflows. To this end we present the structure of data sources and entities, data structures and operators, and their combined use within script programs.

Entities (kind:"dblp_author")		Entities (kind:"dblp_pub")		Mapping	
id	attributes	id	attributes	dblp_author	dblp_pub
A1	name:"A Smith", affiliation:"MIT"	D1	title:"On XML", year:"2000"	A1	D1
A2	name:"B Smith", affiliation:"Google"	D2	title:"Cloud 2.0", year:"2010"	A1	D2
		D3	title:"Matching", year:"2005"	A2	D2
				A2	D3

Fig. 1. Examples for two entities sets (left) of kind dblp_author and dblp_pub, respectively, and a mapping (right)

2.1 Entities and Data Sources

CloudFuice employs the simple and flexible entity-attribute-value model. An entity is of a certain kind and has a kind-specific id, i.e., the pair (kind, id) identifies an entity unambiguously. Each entity has a (possibly empty) list of attributes that can be single-valued (numbers, strings) or multi-valued (i.e., set of single values). This key-value format can be well supported by cloud-based key value stores.

CloudFuice accesses external data sources to obtain entities via three defined methods: **search**, **get**, and **link**. These methods reflect the characteristics of typical web data sources such as (static) web pages, (entity) search engines, and RDF data sources. We therefore assume that it is comparatively easy to implement the following methods (as web services) for a particular data source. Note that not all methods need to be available for all sources.

The **search method** returns entities of a given kind using a specified query. CloudFuice doesn't impose any restrictions on the type of query. The query could be a simple keyword if the source supports keyword search, e.g., a product search engine. If the source supports structured query languages such as SQL or SPARQL, the query could be more complex, e.g., a SQL WHERE condition. The **get method** retrieves all available information for a given entity. This is especially important for frequently changing information, e.g., the number of citations of a publication or the current offered price for a certain product. A possible realization of the get method is a request to an entity specific HTML page (e.g., a product detail page) with subsequent screen scraping. The **link method** retrieves all entities that are connected to a specified entity. This method reflects the nature of web data where the entities are inter-connected by HTML hyperlinks or RDF links in the cloud of linked data. Examples are the list of citing publications for a given publication or the list of products for a specified manufacturer.

2.2 Data Structures and Operators

CloudFuice follows a script programming approach for defining information integration dataflows. It builds up on our iFuice idea [20] and is tailored to web data integration by providing simple and powerful operators. CloudFuice does

not require prior generation of metadata models (global schemas) but lets developers take responsibility to define semantically meaningful integration within their scripts. For example, CloudFuice introduces queries as building blocks and thus enables developers to carefully construct queries in a programmatic way for specific data sources.

A CloudFuice script contains several operator calls like in programs of imperative programming languages. Figure 2 illustrates an example script that will be used throughout the paper and will be discussed in more detail in Section 2.3. The output of any operator can be bound to a variable for later use, e.g., as input to other operators. Script programmers should not be limited to the execution of one source access method at a time but are provided with powerful set-oriented operators. To this end, CloudFuice supports three set-based data structures.

Entities of the same kind A can be subsumed in a set E_A . Figure 1 (left) illustrates two sets of entities. One is a set of DBLP authors, the other a set of DBLP publications. **Mappings** represent directed binary relationships between entities. A mapping $M_{A \times B}$ is a set of correspondences, i.e., pairs of entities of kind A and B , respectively. Figure 1 (right) depicts an example mapping between the DBLP authors and DBLP publications. The author named “A Smith” is linked to the XML and the Cloud paper whereas “B Smith” relates to the Cloud and the Matching paper. Correspondences can be annotated with similarity values or other meta data, e.g., to reflect a confidence level for entity matching [22]. In this paper we restrict the mapping definition to entity pairs without annotations but we present an extension in [23].

Queries are the third type of data structures because they are key to efficient data retrieval from web sources. Providing an explicit data structure for queries Q enables the generation and processing of queries and therefore allows for sophisticated query generation mechanisms [9] that are especially important for entity search engines.

All data structures can be both input and output for operators. CloudFuice provides operators for fetching data from sources and for matching entities. In addition, auxiliary operators are provided for basic data processing. In the following we give a brief overview of selected operators. A complete operator list (incl. signature and definition) can be found in [23].

Source operators fetch data from (web) data sources by employing set-based source access methods. For example, the operators `searchInstances` and `getInstances` take a set of queries or entities, respectively, and call the corresponding source method, `search` or `get`, for every single query/entity. The operator result is then the union of the source methods result. The operators `traverse` and `map` employ the `link` method in a similar way. The difference between these two operators is that `traverse` returns the union of all `link` results whereas `map` incorporates the input entities by returning correspondences between input entities and the resulting `link` entities. **Matching operators** provide techniques for entity matching. The input of a matching operator are two sets of entities and the output is a mapping containing the corresponding, i.e., matching entities. To further restrict the matching, the input can already be a mapping instead of

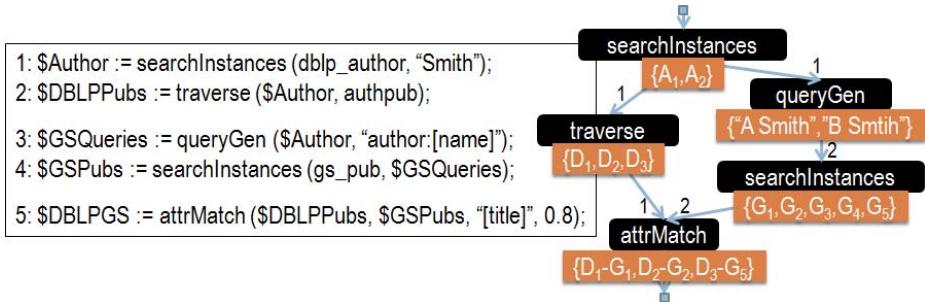


Fig. 2. Left: Script notation of an example dataflow for the integration of publications from DBLP and Google Scholar (GS). **Right:** Graph representation of the example dataflow. The graph also contains example output for each operator, e.g., entities $\{A_1, A_2\}$ for the first `searchInstances` operator.

two input sets. The matching operator then only compares entity pairs of the input mapping. CloudFuice provides a generic attribute matcher `attrMatch` for the common use case of entity matching using attribute similarity and a threshold. In addition, external match algorithms can be plugged in via the `match` operator. **Auxiliary operators** act as the “glue” between operators. Since all data structures are set-based, common set operations `union`, `intersect`, and `diff` can be applied. The `compose` operator allows for composition of mappings, e.g., for input correspondences (a, b) and (b, c) `compose` derives the output correspondence (a, c) . The `filter` operator reduces a set of entities or correspondences using a filter criterion. Finally, `queryGen` allows for a flexible query generation based on entities’ attribute values [9].

2.3 Scripts and Dataflows

Figure 2 (left) shows an example CloudFuice script that integrates publication data from DBLP and Google Scholar. The first script line searches for authors in the DBLP data source and binds the resulting entities to the variable `$Author`. The retrieved authors are input to a `traverse` operator (line #2) that retrieves the corresponding DBLP publications D_1 , D_2 , and D_3 . The author entities are also provided to the `queryGen` operator (line #3) that generates queries using the authors’s names. In the example, the two authors A_1 and A_2 generate one query each and, thus, two queries (“A Smith” for A_1 and “B Smith” for A_2) are bound to the `$GSQueries` variable and sent to the subsequent `searchInstances` operator (line #4). The `searchInstances` operator executes both queries and returns a merged set of Google Scholar publications G_1 to G_5 . Finally, the `attrMatch` operator (line #5) takes both the DBLP and the GS publications as input and determines matching publications. In the example script of Figure 2, two publications are considered to match if they have a title similarity greater than or equal to 0.8. The result is then stored in the variable named `$DBLPGS`.

CloudFuice scripts define a **dataflow**, i.e., an acyclic directed graph with operators as nodes and edges that connect the output of an operator to an input parameter of another operator. Figure 2 (right) shows the dataflow graph for the example script. Each incoming edge is annotated with a parameter index to map the operator output to the specified input parameter. For example, the parameter indexes of two incoming edges for `attrMatch` in Figure 2 are 1 and 2, respectively, to indicate that the `traverse` output is the first parameter whereas the `searchInstances` output is the second parameter of `attrMatch`. Operator parameters that are not output of other operators, e.g., the similarity threshold of `attrMatch`, are considered to be available anytime and we therefore do not include them in the dataflow graph. The dataflow graph is the foundation for CloudFuice’s execution approach that we will explain in the next section.

3 Dataflow Execution

We introduce CloudFuice’s approach to dataflow execution and the application of inter- and intra-operator parallelism. Furthermore, we present the full execution plan for our running example.

3.1 Dataflow Execution Approach

Operators of CloudFuice dataflows are executed within one or multiple operator-specific tasks that can be run in parallel on distributed cloud servers. Tasks may concurrently store their results in a distributed datastore. Furthermore, tasks can be executed as soon as computing resources are available in the cloud to obtain a minimal overall execution time. Finally, our execution model is based on a dynamic invocation of tasks such that each finished task invokes the generation of all operators following in the dataflow as we will detail below.

To support efficient, parallel execution of dataflows, we support both intra- and inter-operator parallelism similar as in parallel database systems [7]. In addition to pipeline parallelism between adjacent operators, data partitioning is utilized to run independent operators on different data in parallel and to parallelize operators on disjoint data partitions.

However, intra-operator parallelism is subject to **partitionable** input data (i.e., operator parameters) only. A parameter p_k of an n -ary operator op is called partitionable if and only if the following two properties hold: (1) The parameter p_k is a set of entities E , or a mapping M (i.e., set of correspondences), or a set of queries Q . (2) For any complete and disjoint partitioning $p_k = p_{k_1} \cup p_{k_2}$ of p_k holds: $op(p_1, \dots, p_k, \dots, p_n) = op(p_1, \dots, p_{k_1}, \dots, p_n) \cup op(p_1, \dots, p_{k_2}, \dots, p_n)$.

We call an operator blocking if none of its parameters is partitionable. A blocking operator is therefore unable to produce any (partial) result until all input data is available. On the other hand, all parameters (of type E , M , or Q) of non-blocking operators are partitionable. For example, all source operators (e.g., `searchInstances`) are non-blocking. Operators that are neither blocking nor non-blocking are called partially blocking. For example, the `diff` operator computes the difference of two entity sets E_1 and E_2 of the same kind. The first

```

1 FUNCTION executeOp (taskIndex, taskRes) BEGIN      1 FUNCTION executeDataflow () BEGIN
2   IF isBlocked() THEN                            2   FOREACH op IN getStartOps() DO
3     return;                                         3     op.executeOp (0, NULL);
4   END                                              4   END
5   FOR i=1 TO n DO // get current param values    5   END
6     pValue[i] = getCurrentValue(i);                11
7   END                                              12 // consider new data only for partitionable task result
8   // consider new data only for partitionable task result
9   IF (taskIndex>0) AND (isPartitionable(taskIndex)) THEN
10     pValue[taskIndex] = pValue[taskIndex] - taskRes;
11   END
12   // data partitioning and task creation          13 partPValue[][] = partitioning (pValue)
13   partPValue[][] = partitioning (pValue)           14 FOR t=1 TO partPValue.length DO
14     createTask (partPValue[t]);                   15   createTask (partPValue[t]);
15   END                                              16   END
16 END                                              17 END
17 END

```

Fig. 3. Pseudocode for executing dataflows, operators, and tasks

parameter E_1 is partitionable whereas the second parameter E_2 is not. The operator needs to know all entities E_2 that must not appear in the operator result before returning any (partial) results. The Appendix of [23] specifies the type for each operator.

The advantages of partitionable input data for an operator are twofold. First, an operator can partition each partitionable parameter and thereby split the operator execution into multiple tasks that can be executed in parallel (intra-operator parallelism). The complete operator result can later be reconstructed from all task results. On the other hand, data partitioning can also be used for asynchronous (pipelined) operator execution. Individual task results, i.e., partial operator results, can already be handed off (or “pushed”) to its succeeding operator (which in turn may already produce a partial result) if the corresponding input parameter of the succeeding operator is partitionable, too. This method therefore allows for an overlapping execution of neighboring operators in the dataflow graph. Finally, independent operators, i.e., operators that neither directly nor indirectly rely on each others’ outputs, are run in parallel (inter-operator parallelism).

Dataflow execution thus entails the correct transformation of a dataflow into a series of independently executable tasks. Figure 3 shows the pseudo code for task generation (methods for dataflow and operator execution) and task execution. Initially the tasks for all start operators, i.e., operators that do not rely on input of other operators, are generated and executed (see method `executeDataflow`). All other operators will be dynamically invoked by the tasks creating their input parameters (see method `executeTask`). The input-generating task then hands over its complete result (`taskRes`) as well as the associated parameter index (`taskIndex`) as parameters to the `executeOp` call. For the invocation of a start operator that does not depend on input data, we use a `taskIndex` value of 0 (see `executeDataflow`).

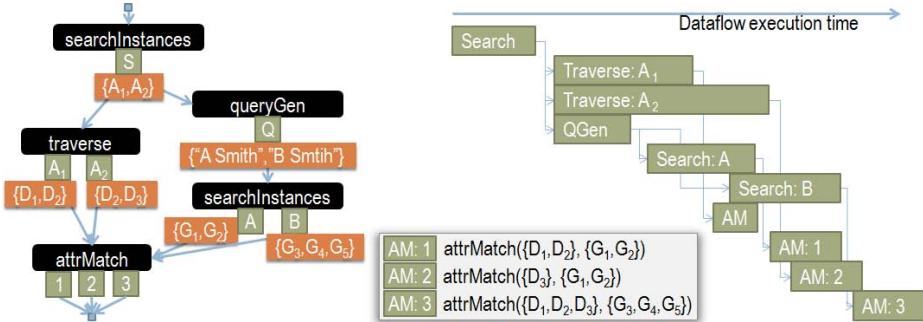


Fig. 4. Example execution of the dataflow depicted in Figure 2. In addition to Figure 2 the dataflow graph also shows the generated tasks (green quadrats below the operator box). The right part shows a timeline of the executed tasks.

The `executeOp` method partitions the input data based on an operator-specific partitioning function `partitioning` and creates a corresponding task for each partition. CloudFuice thereby allows tailored partitioning strategies for operators (see Section 5 for a discussion and evaluation). The task execution first employs the operator-specific computation (`compute`) to achieve the task result which is then stored in a distributed data store. Afterwards all following operators are called (with the task result and the parameter index) which in turn may eventually create new tasks.

The methods for operator and task execution make use of a few auxiliary functions. The function `isBlocked` returns `true` if there is at least one non-partitionable input parameter p_i with incomplete data, i.e., the corresponding input operator has not finished yet, and thus the operator cannot be executed yet. It makes use of `isPartitionable(i)` which is `true` if the parameter p_i is partitionable. `getCurrentValue(i)` reads the current value of parameter p_i from the datastore. The function `getStartOps` returns all start operators of the dataflow and `getNextOps` returns the following operators for a task. With the help of `getIndex(nextOp)` the parameter index for which the current task provides input to operator `nextOp` is determined. Finally, the function `createTask` generates a new task that will eventually be executed by the runtime infrastructure (see Figure 5).

The approach for asynchronous executions of operators/tasks also allows for notifications when an operator is finished and, thus, when the value of the bounded script variable is available. An operator is finished if all of its input operators are finished and there are no unfinished operator tasks. To this end each operator keeps track of the number of generated tasks as well as the number of finished tasks. Finally, a script is finished if all operators are finished. This additional functionality is not shown in Figure 3 in favor of readability.

3.2 Execution Example

Finally, we demonstrate how the example script of Figure 2 will be executed. Figure 4 illustrates the dataflow along with the generated tasks and their

execution time frame. The only start operator, `searchInstances`, is invoked for one query at the beginning and as a result it generates one `search` task. Once finished, it triggers the two subsequent operators `traverse` and `queryGen`. They can run in parallel because they are independent from each other (inter-operator parallelism). The `traverse` operator makes use of intra-operator parallelism and generates two tasks because it is given two input entities (DBLP authors A_1 and A_2). The example assumes that there is no access restriction to the data source so that both tasks can be executed immediately. The `queryGen` operator is also invoked for A_1 and A_2 but generates only one task that emits two queries. This task – once finished – invokes the second `searchInstances` operator. The two input queries lead to two `search` tasks. In this example we assume that there is a source access quota so that the first task can be executed immediately whereas the second task is scheduled with consideration of a reasonable waiting time (handled by a task scheduler, see Section 4).

The last operator, `attrMatch`, takes as input the output of the two operators `traverse` and `searchInstances`. The operator input data comes in four data chunks because both input operators employ two tasks each for their execution. The order of the incoming data chunks is arbitrary and in the example of Figure 4 we assume the following order: `traverse:A1` ($= \{D_1, D_2\}$), `search:A` ($= \{G_1, G_2\}$), `traverse:A2` ($= \{D_2, D_3\}$), and `search:B` ($= \{G_3, G_4, G_5\}$). Once finished, each task invokes the `attrMatch` operator.

In our example we consider a non-blocking implementation of `attrMatch`, i.e., `attrMatch` process all pairs of the Cartesian product independently, e.g., with the help of common string similarity measures such as Edit Distance. It is therefore amenable to asynchronous execution, i.e., it can produce partial results on partial input data. However, the first operator call does not yet create any tasks because at this point there are only DBLP publications available but no GS publications. The second operator call generates the first task `AM:1` that matches all available DBLP entities with the new available GS entities (see the box in the middle of Figure 4). The third operator call completes the DBLP input entities and creates task `AM:2`. This task process the new DBLP entities (only D_3 is new because D_1 and D_2 have been already retrieved by the previous `traverse` task) along with all available GS entities. Finally, the second query of `searchInstances` is finished and the corresponding task calls `attrMatch` for the fourth time. To this end, `attrMatch` creates task `AM:3` that takes as input all available DBLP entities and the newly added GS entities $\{G_3, G_4, G_5\}$. The box in the center of Figure 4 summarizes the three employed `attrMatch` tasks along with their input data. It thereby illustrates that the `attrMatch` operator indeed processes the entire cross product of all DBLP and GS entities. In the example of Figure 4 the partitioning is solely driven by the outcome and the execution order of the input operator tasks. However, large entity sets may require an additional partitioning to reduce the workload per task (see Section 5).

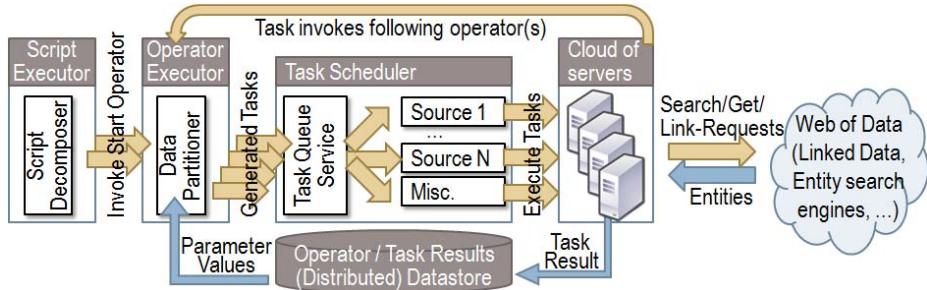


Fig. 5. Architecture of the CloudFuice approach. It employs a distributed data store, a task queue service, and a cloud of servers for task execution.

4 Web-Based Architecture and Prototype

Figure 5 depicts the overall CloudFuice architecture. It comprises a script and operator executor, a task scheduler, and an elastic cloud of servers. The script and operator executor realize CloudFuice’s execution approach. A given script is first converted into a dataflow and then decomposed into a set of operator calls which are in turn transformed into multiple tasks. Tasks are subject to a scheduling mechanism that takes into account access restrictions of external sources. Tasks are executed on a cloud of servers that are capable of (parallel) executing tasks as web services. All tasks store their results in a distributed data store and invoke other operators if necessary. If the last task of an operator has finished, it may notify external applications about the availability of an operator result (not shown in Figure 5).

The script executor analyzes a given CloudFuice script and decomposes it into a data flow. The script executor then invokes all start operators, i.e., all operators that do not rely on input of other operators. The operator executor retrieves the relevant parameter values from the datastore. If the operator can be executed, the executor applies an operator-specific partitioning strategy on the input data and generates one or multiple tasks. Each task is assigned the relevant partition of the operator’s input data as well as the list of following operators. Tasks are sent to the task scheduler.

The task scheduler provides a task queue for each source and a task is appended to a task queue if it is supposed to send a request to the corresponding source. Each task queue employs a simple bucket algorithm for scheduling task execution because source access is typically limited by quotas and exceeded quotas may cause requests failures. The bucket size determines how many tasks can be executed in parallel, i.e., how many requests can be sent to the source simultaneously. The execution rate controls the average number of task executions for a time window, e.g., 1 task (request) per second. The scheduler also provides an additional unlimited task queue (“miscellaneous”) for all other tasks. Since tasks may fail due to several reasons (e.g., network congestion or server unavailability) each task queue also provides a retry mechanism for failed tasks.

The CloudFuice architecture realizes task execution by web service requests. Requests can be handled by different servers (nodes) that all have access to a distributed data store for storing task results. The actual task implementations are encapsulated as web services and can, thus, be realized in virtually any programming language. Each task may request data from the Web using one of the source access methods (`search`, `get`, or `link`). Note that the task implementation does not need to deal with any source constraints because this is already handled by the task scheduler. Moreover, no global coordinator is needed for the task execution because each task knows what operator is supposed to be invoked after its completion.

The current prototype implementation employs Google App Engine, a platform for running web applications on multiple nodes. New server instances become automatically (un)available based on the number of tasks. All operators and tasks are implemented as REST-based web services and use JSON as data exchange format. The prototype employs Google App Engine's datastore that implements the Bigtable data model [4]. Task results can be concurrently stored using a unique task id which is assigned during task generation. Each task result contains a reference to the corresponding operator and, thus, operator results can be retrieved by merging all corresponding task results. The prototype employs Google Spreadsheet as script development environment (see [23] for screenshots). The spreadsheet contains both CloudFuice scripts and data and, thus, will serve as an integrated development environment. Google Spreadsheet executes CloudFuice scripts and retrieves data by HTTP requests. On the other hand, Google Spreadsheet's API enables the CloudFuice server to directly update results in a spreadsheet (data pushing) even if the spreadsheet is closed. Google Apps Script is used to implement simple user interfaces and execute parametrized CloudFuice scripts. This mechanism acts as a simple way for mashup development.

5 Evaluation

We evaluate the practicability of our approach with the help of our prototype and thereby demonstrate that the use of cloud technologies can improve the runtime performance of CloudFuice scripts. In particular we will examine partitioning strategies for entity matching and the effect of asynchronous script execution.

The first experiment deals with effective and efficient data partitioning which is key to a correct and efficient execution of dataflows. Recall that the execution approach ensures that only partitionable data is subject to intra-operator parallelism and that different partitioning strategies can be applied. Fine-grained partitioning strategies generate small tasks that can be executed in parallel but suffer from the additional overhead of task creation, scheduling, and execution. Furthermore, danger of skew effects also grows with more tasks since the slowest task determines the overall execution time. On the other hand coarse-grained strategies result in few large tasks that may not fully exploit the power of the available computing resources. For intra-operator parallelism, we evaluate size-based partitioning functions for entity matching similar to [13]. A maximal block size b ensures that each match task only process a limited part of the Cartesian

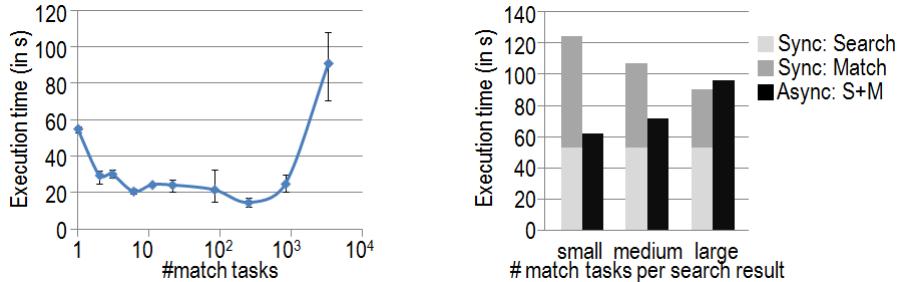


Fig. 6. Evaluation of partitioning strategies for entity matching (left) and the influence of asynchronous dataflow execution (right)

product, i.e., at most $b \times b$ entities per task. For our experiment we employ attrMatch with two entity sets of size $|R| = 471$ and $|S| = 16,269$, respectively. Each set is evenly divided into blocks of maximal size b and one task is generated for each pair of blocks. For example, a block size $b = 100$ leads to 5 blocks for R and 163 blocks for S and thus $5 \cdot 163 = 815$ match tasks are created.

Figure 6 (left) shows the measured average script execution time (average over three runs, minimal and maximal values are shown as error bars) for different block sizes. The x -axis denotes the resulting number of tasks. The execution time can be significantly reduced from 55 seconds (computation is realized by one task only) to 14 seconds if approx. 250 tasks are employed. However, if the number of tasks is increased even further the execution time is overwhelmed by the additional task management overhead. Note that the prototype runs on Google App Engine that does not allow configuration of computing capacities, i.e., creation and utilization of node instances cannot be controlled. Instances become instead available and unavailable, respectively, by an internal heuristics based on the current number of tasks.

In a second experiment we demonstrate the influence of asynchronous script execution (inter-operator parallelism). The evaluation scripts contains two steps. First, 1,180 entities are requested by 49 queries (`searchInstances`). We assume a reasonable source quota of 1 query per second, i.e., query execution process takes about 50 seconds. In a second step the obtained entities are matched against a given set of 16,269 entities like in our previous experiment. As shown in our running example (see Figure 4) both input sets of attrMatch are partitionable, i.e., entity matching can already process partial input data. For a synchronous execution we assume that none of the attrMatch parameters are partitionable and thus the operator cannot start until all input data becomes fully available. Synchronous execution behavior of attrMatch is achieved by modifying the `isBlocked` function accordingly (see pseudo code in Figure 3).

Figure 6 (right) compares the overall times for synchronous vs. asynchronous execution using different partitioning strategies for attrMatch. The synchronous execution is composed of the search time and matching time. The search time remains the same for all partitioning strategies because it is dominated by the source quota. On the other hand, an increasing number of tasks decreases the

time for entity matching similar to what we have observed in Figure 6 (left). Figure 6 (right) proves that CloudFuice’s asynchronous dataflow execution can significantly reduce the execution time by an early hand-off of partial results. The source access is, of course, a lower bound but the asynchronous model already performs entity matching while still querying the data source. It thereby reduces the remaining workload after the last query result becomes available. However, the asynchronous model is characterized by an increasing execution time for an increasing task number. This is due to the fact that each search result retrieves a comparatively small number of entities (≈ 24) which then have to be matched against the large set of 16,269 entities. Obviously a fine-grained partitioning is not beneficial for such imbalanced match tasks. For example, if we assume that each search result retrieves at most 100 entities a block size of $b = 100$ would result in 163 tasks per search result. The overall number of match tasks is therefore $49 \text{ (search queries)} \times 163 = 7,987$ that deteriorates execution time.

In general, CloudFuice’s execution model has to combine a partitioning strategy with an incremental availability of input data due to asynchronous execution. This is especially important against the background of an additional overhead for task generation, scheduling, and transferring. We will therefore investigate in adaptive partitioning strategies in future work. For example, a partitioning strategy may not create any tasks until a minimal number of entities is available when dealing with partial results. Furthermore, operators that do not have following operators might apply a different partitioning strategy because there is no benefit in forwarding partial results.

6 Related Work

Mashup-based data integration has become very popular in recent years and many tools and frameworks have been developed [17]. Applications need components for data, process, and presentation level [18] and CloudFuice mainly focuses on the data level. A popular approach are pipes, e.g., as used in Yahoo! Pipes, Damia [21], or [15], that process entity sets via relatively simple user-specified dataflows. These tools have demonstrated to be applicable in different settings since they offer a powerful and easy-to-use interface for inexperienced users. However, they are not designed for more advanced data integration problems that have to deal with dirty data and, thus, require advanced operators. For example, entity matching and query generation are key to achieve accurate and complete integrated results. This, on the other hand, requires more advanced skills in mashup development. CloudFuice therefore strives for a good balance between powerful data integration operators and a simple scripting language.

A few recent mashup platforms also deal with mashup efficiency. CoMaP [10] targets a distributed mashup execution that minimizes the overall mashup execution time of multiple hosted mashups. It is based on a general mashup dataflow model with operators that can be executed on different nodes. A dynamic scheduling takes into account several parameters such as network and users. The AMMORE [11] system even modifies original mashup dataflows to avoid duplicate computations and unnecessary data retrievals. To this end, AMMORE identifies

common operator sequences in different mashups and executes them together. CoMap, AMMORE, and CloudFuice share the same goal of efficient dataflow execution but have slightly different focuses. CloudFuice does not deal with multiple mashups and users but task scheduling is driven by data source constraints and available computing capacity instead of usage or network traffic.

The recent shift in web infrastructures from high-end server systems to clusters of commodity hardware (also known as cloud) has triggered research in parallel data processing in a wide variety of applications. Driving forces are simple and powerful parallel programming models such as MapReduce [6] and Dryad [12]. Although the MapReduce program model is limited to two dataflow primitives (map and reduce), it has proven to be very powerful for a wide range of applications. On the other hand Dryad supports general dataflow graphs. Freely available frameworks such as Hadoop further stimulate the popularity of distributed data processing. Higher-level languages can be layered on top of these infrastructures. Examples include the high-level dataflow language Pig Latin [19], Nephele/PACTs [1] (based on MapReduce), DryadLINQ [24] as well as SCOPE [3] that offers a SQL-like scripting language on top of Microsoft's distributed computing platform Cosmos. Similar to Cloudfuice, a high-level program is decomposed into small buildings blocks (tasks) that are transparently executed in a distributed environment. On the other hand, CloudFuice targets the fast development of dataflows for web applications and therefore relies on a common web engineering tools. Furthermore CloudFuice deals with asynchronous dataflow execution, e.g., due to continuous data input from query results, in contrast to synchronized offline data analysing/processing. For example, MapReduce enforces synchronization between the map and reduce phase (a disadvantage that has been recently addressed in [8]).

7 Conclusions and Future Work

We presented CloudFuice, a flexible system for specification and execution of dataflows for data integration. The task-based execution approach allows for an efficient, asynchronous, and parallel execution within the cloud and is tailored to recent cloud-based web engineering instruments. We have demonstrated CloudFuice's applicability for mashup-based data integration in the cloud with the help of a first prototype implementation.

Future work includes the development of adaptive partitioning strategies as well as the further development toward a comprehensive mashup platform. We will also investigate how dataflows can be automatically generated from declarative queries.

References

1. Battré, Ewen, Hueske, Kao, Markl, Warneke: Nephele/PACTs: a programming model and execution framework for web-scale analytical processing. In: SoCC (2010)
2. Bizer, Heath, Berners-Lee: Linked data - the story so far. IJSWIS 5(3) (2009)

3. Chaiken, Jenkins, Larson, Ramsey, Shakib, Weaver, Zhou: SCOPE: Easy and efficient parallel processing of massive data sets. In: PVLDB, vol. 1(2) (2008)
4. Chang, Dean, Ghemawat, Hsieh, Wallach, Burrows, Chandra, Fikes, Gruber: Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst. 26 (2008)
5. Cheng, Yan, Chang: Entityrank: Searching entities directly and holistically. In: VLDB (2007)
6. Dean, Ghemawat: MapReduce: Simplified data processing on large clusters. Communications of the ACM 51(1) (2008)
7. DeWitt, Gray: Parallel database systems: The future of high performance database systems. Communications of the ACM 35(6) (1992)
8. Elteir, Lin, Feng: Enhancing mapreduce via asynchronous data processing. In: IC-PADS (2010)
9. Endrullis, Thor, Rahm: Evaluation of Query Generators for Entity Search Engines. In: USEITIM (2009)
10. Hassan, Ramaswamy, Miller: Comap: A cooperative overlay-based mashup platform. In: CoopIS (2010)
11. Hassan, Ramaswamy, Miller: Enhancing Scalability and Performance of Mashups Through Merging and Operator Reordering. In: ICWS (2010)
12. Isard, Budiu, Yu, Birrell, Fetterly: Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In: EuroSys Conference (2007)
13. Kirsten, Kolb, Hartung, Gross, Köpcke, Rahm: Data Partitioning for Parallel Entity Matching. In: QDB (2010)
14. Köpcke, Rahm: Frameworks for entity matching: A comparison. Data Knowl. Eng. 69(2) (2010)
15. Le-Phuoc, Polleres, Hauswirth, Tummarello, Morbidoni: Rapid Prototyping of semantic Mash-ups through semantic Web Pipes. In: WWW (2009)
16. Lenzerini: Data integration: A theoretical perspective. In: PODS (2002)
17. Lorenzo, Hacid, Paik, Benatallah: Data Integration in Mashups. SIGMOD Rec. 38 (2009)
18. Maximilien, E.M., Wilkinson, H., Desai, N., Tai, S.: A Domain-Specific Language for Web APIs and Services Mashups. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 13–26. Springer, Heidelberg (2007)
19. Olston, Reed, Srivastava, Kumar, Tomkins: Pig Latin: A Not-So-Foreign Language for Data Processing. In: SIGMOD (2008)
20. Rahm, Thor, Aumueller, Do, Golovin, Kirsten: iFuice - Information Fusion utilizing Instance Correspondences and Peer Mappings. In: WebDB (2005)
21. Simmen, Altinel, Markl, Padmanabhan, Singh: Damia: Data Mashups for Intranet Applications. In: SIGMOD (2008)
22. Thor, Rahm: MOMA - A Mapping-based Object Matching System. In: CIDR (2007)
23. Thor, Rahm: CloudFuice: A flexible Cloud-based Data Integration Approach. Technical report, University of Leipzig (2011), <http://dbs.uni-leipzig.de/publication/year/2011>
24. Yu, Isard, Fetterly, Budiu, Erlingsson, Gunda, Currey: Dryadlinq: a system for general-purpose distributed data-parallel computing using a high-level language. In: OSDI (2008)

Bootstrapping Trust of Web Services through Behavior Observation

Hamdi Yahyaoui¹ and Sami Zhioua²

¹ Computer Science Department, Kuwait University
P.O. Box 5969, Safat 13060, State of Kuwait
hamdi@sci.kuniv.edu.kw

² Information and Computer Sciences Department, KFUPM
P.O. Box 958, Dhahran 31261, KSA
zhioua@kfupm.edu.sa

Abstract. We present in this paper a new Web services trust bootstrapping technique, which consists in observing several interactions of a user with a Web service. The obtained observations sequence is modeled as a Hidden Markov Model (HMM) and matched against pre-defined trust patterns in order to assess the behavior of the Web service under observation. The pre-defined trust patterns are specifications of possible behaviors of Web services. Based on the matching result, an initial trust value is assigned to the observed Web service. Our experimental results show that our technique has good precision and recall rates together with a fair distribution of trust values.

Keywords: Trust, Web services, Bootstrapping, Hidden Markov Model.

1 Introduction

Web services are becoming nowadays a powerful technology which allows users to invoke, in a transparent way, distant services through standard Web protocols. With a variety of functionalities, plenty of Web services are deployed in the internet. Consumers judge the performance of these services based on their non-functional quality attributes such as response time, availability, throughput, etc. Such judgement reflects how much a Web service is trusted for performing a certain task in the expected way. The non-functional quality attributes can be leveraged to derive a trust value for a Web service as suggested in [14]. Trust can be used as an indicator for the possible future behavior of a Web service. One of the important issues in establishing the trust for Web services is the assignment of a trust value to unknown Web services. This is known as the bootstrapping issue. Bootstrapping is even more challenging when there is an absence of user recommendations. Unfortunately, this issue did not receive much attention in the related work as will be shown later, while in reality it has an important impact on the robustness of the elaborated trust model. The aim of our work is to devise a new bootstrapping technique based on trust patterns and Hidden Markov Model (HMM) in the absence of recommendations. The proposed technique focus on the recognition of the behavior of a Web service rather than deriving a trust value from a simple interaction.

The rest of the paper is as follows. In section 2, we recall some definitions about HMM. Section 3 is devoted to the review of published bootstrapping techniques. Section 4 is dedicated to the presentation of our bootstrapping technique. In section 5, we provide experimental results and analysis. Finally, some conclusions and future works are drawn in Section 6.

2 Background

This section is devoted to the presentation of some definitions that are useful to have a clear understanding of our technique.

2.1 HMM Definition

A Hidden Markov model (HMM) [11] is a Markov chain where states are not completely observable. That is, the underlying mechanics corresponds to a Markov chain but the state of the system at a given moment is not exactly known. Instead, at each time step, a stochastic observation is generated based on the current state.

Definition 1. *Hidden Markov Model*

A HMM is a tuple $(\mathcal{S}, T, \mathcal{O}, Q, \pi)$ where

- \mathcal{S} is a set of N states $\{s_1, s_2, \dots, s_N\}$. Besides, s^t refers to the state at time step t . Note the difference between the subscript and superscript versions s_1 and s^t .
- $T : \mathcal{S} \rightarrow \Pi(\mathcal{S})$ is a state transition function which maps each state \mathcal{S} to a probability distribution over \mathcal{S} . $T_{s_i \rightarrow s_j}$ denotes the probability of making a transition from s_i to s_j .
- \mathcal{O} is a set of M observations $\{o_1, o_2, \dots, o_M\}$. Similarly to states, let o^t be the observation at time step t .
- $Q : \mathcal{S} \rightarrow \Pi(\mathcal{O})$ is an observation function. Q_s^o denotes the probability of observing o while in state s .
- π is the initial state distribution, with $\pi(s)$ denotes the probability of s being the initial state.

For simplicity, we use the compact notation $\mathcal{H} = (T, Q, \pi)$.

2.2 Probability of Accepting an Observation Sequence

One of the basic problems with HMMs is how to compute the probability of an observation sequence given a HMM model. Let $O = o^1 o^2 \dots o^n$ be an observation sequence of length n and let $\mathcal{H} = (T, Q, \pi)$ be an HMM model. The goal is to compute the conditional probability $P(O|\mathcal{H})$.

The sequence O can be generated from \mathcal{H} using different state sequences of length n . Hence, one way of computing $P(O|\mathcal{H})$ is to sum over all possible state sequences of length n . Let $S = s^1 s^2 \dots s^n$ be such a sequence. The probability that O is generated

using S is the probability of getting observation o^1 from state s^1 , o^2 from s^2 and so on until o^n from s^n , that is,

$$P(O|S, \mathcal{H}) = Q_{s^1}^{o^1} Q_{s^2}^{o^2} \dots Q_{s^n}^{o^n} \quad (1)$$

However, to sum over all possible state sequences, one needs instead the joint probability of O and S , namely, $P(O, S|\mathcal{H})$. By the Bayes theorem we have:

$$P(O, S|\mathcal{H}) = P(O|S, \mathcal{H})P(S|\mathcal{H}) \quad (2)$$

Where $P(S|\mathcal{H})$ is the probability of going through the state sequence S :

$$P(S|\mathcal{H}) = \pi(s^1)T_{s^1 \rightarrow s^2}T_{s^2 \rightarrow s^3} \dots T_{s^{n-1} \rightarrow s^n} \quad (3)$$

We have now all the ingredients to illustrate the detailed computation of $P(O|\mathcal{H})$:

$$\begin{aligned} P(O|\mathcal{H}) &= \sum_{S=s^1 s^2 \dots s^n} P(O|S, \mathcal{H})P(S|\mathcal{H}) \\ &= \sum_{S=s^1 s^2 \dots s^n} \pi(s^1)Q_{s^1}^{o^1} T_{s^1 \rightarrow s^2} Q_{s^2}^{o^2} \\ &\quad \dots T_{s^{n-1} \rightarrow s^n} Q_{s^n}^{o^n} \end{aligned} \quad (4)$$

Equation (4) is clearly not easy to compute. For one reason, it sums over all possible state sequences whose number is in the range of N^n . Adding the fact that for each such sequence $2n$ operations are performed, the total number of computations required to compute $P(O|\mathcal{H})$ is in the order of $2nN^n$. Hence, Equation (4) is not an efficient way of computing $P(O|\mathcal{H})$. A better approach is what is known as Forward-Backward procedure [1].

The Forward-Backward approach to compute $P(O|\mathcal{H})$ is based on defining the forward probability

$$f_t(i) = P(O, s_i|\mathcal{H}) \quad (5)$$

$f_t(i)$ is the partial probability of observing the sequence $O = o^1 o^2 \dots o^t$ and end-up in state $s^t = s_i$. The key point is that $f_t(i)$ can be defined inductively as follows:

$$f_t(i) = \begin{cases} \pi(s_i)Q_{s_i}^{o^t} & \text{if } t = 1 \\ \sum_{j=1}^N (f_{t-1}(j)T_{s_j \rightarrow s_i})Q_{s_i}^{o^t} & \text{if } 1 < t \leq n \end{cases}$$

$P(O|\mathcal{H})$ can be expressed in terms of the forward probability simply as:

$$P(O|\mathcal{H}) = \sum_{i=1}^N f_n(i) \quad (6)$$

The Forward-Backward approach to compute $P(O|\mathcal{H})$ requires on the order of N^2n computations, which is clearly better than the $2nN^n$ of Equation (4). The key point is that since there are only N states, all the possible state sequences will remerge into these N states, no matter how long the observation sequence.

3 Related Work

Bootstrapping a new element in a system means assigning an initial trust/reputation value for it. Such value has an impact on the security of the whole system since there is no prior knowledge about the possible behavior of the new element. Henceforth, bootstrapping is paramount for the security of software and systems. The bootstrapping issue was studied in few research initiatives. In what follows, we present these initiatives and pinpoint their pros and cons. It is worth to mention that the terms reputation and trust have different meanings. Reputation denotes a group opinion about a peer while the trust denotes an individual opinion.

3.1 Default Value Technique

The default value technique was designed for peers collaborating in a Mobile Ad hoc Network (MANET). In this technique, the new peers that join the network get a default reputation value [7]. The value is generally considered as a threshold under which a peer is considered as malicious. One disadvantage of this technique is that, depending on the default initial value, it can either favor existing peers or new peers that join the network. If the initial reputation is high, existing peers are disadvantaged, since the new peer is assigned a value that can be higher than existing peers that collaborated a lot and strived to have a good reputation. This encourages malicious peers to leave and join again the network with new identities to avoid their bad reputation (this is known as white-washing). A low reputation value will discourage new peers from being involved in collaborations.

3.2 Punishing Technique

The punishing technique [2] is proposed as a solution to overcome the white-washing issue. By giving a low reputation value to the new peer, it ensures that a malicious peer that is trying to leave and join again the network will not gain that much. However, a very low value will make the new peers disadvantaged and perhaps no existing peer will collaborate with them, which makes them isolated in the network.

3.3 Adaptive Technique

The adaptive technique is part of a trust model for Web services, which is proposed in [5]. In this technique, a new Web service which is willing to collaborate is assigned a reputation value that depends on the rate of maliciousness, which is defined as the ratio of the number of collaborations (considered as transactions in that work) where the Web services defect, to the total number of collaborations. This requires tracking each collaboration quality. A Web service rater marks collaboration as acceptable if the Web service did collaborate as expected and defective in the opposite case. An unknown Web service is assigned a high initial reputation value when the rate of maliciousness is low and a low initial reputation value when that rate is high. The issue with this technique is that a leaving malicious Web service can try to rejoin the network with a new URL and get a better reputation value than his old value if he leaves the network with a low reputation value while the maliciousness rate is very low at that moment.

3.4 Prediction Technique

In [4], the author proposed a HMM-based prediction model to assess a provider reputation when adequate number of rater recommendations is not available. A service consumers HMM is trained using the recommendations provided by some evaluators. Once a reliable model is developed, the high and low reputations of the services are predicted. In the next step, the service consumer compares all the predicted provider reputations. The provider which has the highest predicted reputation for the next time instance is chosen for interaction. After each interaction, the observed behavior values and present recommendations are input to the HMM for the sake of refining the model. This model is used for predicting a single reputation value for a Web service and can't be used to classify a Web service behavior. Such classification is needed since at one time instance a service reputation value can't reveal accurately its long term behavior. Besides, it is very often to have a lack of recommendations about a Web service which is unknown to raters. Hence, assuming always the existence of recommendations is not always a realistic assumption.

4 A New Pattern-Based Bootstrapping Technique

The techniques mentioned in the previous section focus on a local evaluation of the reputation level of an unknown peer or Web service. This cannot provide an accurate assessment of the trustworthiness of a Web service. We follow a global strategy where a Web service is evaluated during a certain time frame in which a sequence of trust observations is built. Based on this sequence, we are able to judge, whenever possible, the kind of behavior a Web service has, i.e., the category to which a Web service belongs (trusted, malicious, redemptive, etc.). More precisely, our technique consists in first evaluating a Web service during a certain period of time. During that period, a sequence of trust observations is built. Each trust observation (T for Trusted and U for Untrusted) denotes the degree of compliance of the non-functional quality attributes of a Web service (e.g. response time, availability, reliability, etc.) with the announced quality attributes (as part of Service Level Agreement). The obtained sequence of trust observations is then matched against pre-defined trust patterns. The objective of the matching step is to have a global judgement about the behavior of a Web service and derive a trust value based on the recognized behavior. In what follows, we provide the ingredients of our technique.

4.1 Trust Patterns

In this section, we define trust patterns which refer to particular Web service behaviors. A trust pattern is a sequence of trust observations from which a clear conclusion about the behavior of a Web service can be drawn. We provide a formal characterization of trust pattern categories:

- Trustworthy: T^+
- Oscillating: $(T^+.U^+.T^+.U^+)^*$ Or $(U^+.T^+.U^+.T^+)^*$
- Malicious: U^+

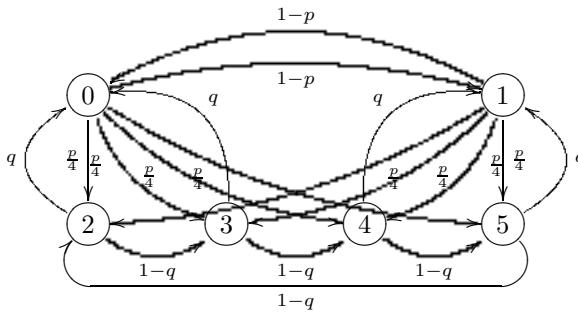
- Betraying: $(T^+.U^+)^*$
- Redemptive: $(U^+.T^+)^*$

Where T and U denote respectively a Trusted/Untrusted observation, while the operators $^+$ and $*$ denote respectively a non-empty sequence and the repetition of a sequence. A trust observation is generated after one interaction with a Web service while a sequence of trust observations is constructed during a certain time frame. We restrict the description of patterns to the aforementioned categories but the extension of these categories is possible.

The sequence notation for a trust pattern is used for readability aims. However, we use another notation for specifying such patterns. Indeed, a trust pattern can be represented using a HMM. The advantage of this representation is that a HMM is more expressive than a static sequence of trust observations. Indeed, a HMM is more compact and it allows to represent a range of trust observation sequences falling in the same category. For example, consider the two following trust observation sequences which clearly fall in the category of an oscillating pattern:

1. T.U.T.U.T.U.T.U.T.U
2. T.T.U.U.T.T.U.U.T.T.U.U

The two trust observation sequences can be represented by a single HMM of the form:



Where states 0, 2 and 3 generate always the trust observation T and the states 1, 4 and 5 generate always the trust observation U . Another advantage of using HMMs in specifying trust patterns is that a HMM can capture patterns resulting from alternating between the two patterns above.

The five HMMs are trained given a single or multiple trust observation sequences. The training process is inspired by the seminal work of Rabiner[11]. Given a trust observations sequence $O = o^1o^2 \dots o^n$, the training aim is to find how to adjust the model parameters $\mathcal{H} = (T, Q, \pi)$ to maximize $P(O|\mathcal{H})$. There is no optimal way for estimating the model parameters so that to maximize the probability of acceptance of a trust observations sequence. The best one can do is to choose the model parameters $\mathcal{H} = (T, Q, \pi)$ such that $P(O|\mathcal{H})$ is locally maximized using an iterative procedure such as the Baum-Welch method [1].

4.2 Trust Observation Generation

A trust observation (T or U) is generated during an evaluation period through the assessment of quality attributes of a Web service. Each quality attribute (such as response

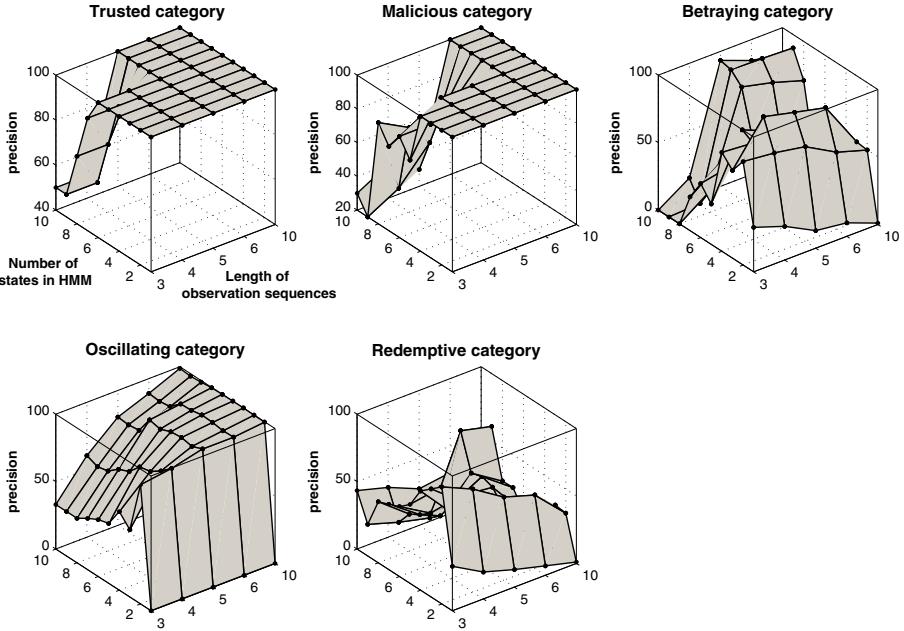


Fig. 1. The precision of classifying a bank of observation sequences as we increase the number of states of the HMM (y-axis) and the length of the observation sequences (x-axis)

time, availability, etc.) has a certain expected and actual value. The expected values are announced by Web service providers through a Service Level Agreement (SLA) with consumers. The actual values are computed during the evaluation period and matched against the expected ones. To assess the trustworthiness of a Web service during one interaction, we compute the Root Mean Square Error (*RMSE*):

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (v_i - v'_i)^2}{n}} \quad (7)$$

Where v_i denotes the actual value of a quality attribute q_i , v'_i the expected value of a quality attribute q_i and n the total number of assessed quality attributes. We assume that all values v_i and v'_i are normalized before computing *RMSE*. This means that *RMSE* has a value between 0 and 1. During an interaction i , a trust observation o^i is generated based on the following equation:

$$o^i = \begin{cases} T & \text{if } RMSE < \xi \\ U & \text{Otherwise} \end{cases} \quad (8)$$

Where ξ denotes a tunable threshold between 0 and 1. It reflects the maximal marginal error that a Web service is allowed to have and over which it is judged as not trusted during the evaluation of an interaction.

4.3 Bootstrapping Trust of Web Services Based on Trust Patterns

We present hereafter an algorithm for bootstrapping the trust of Web services based on matching a trust observations sequence with pre-defined trust patterns (Algorithm 1). The algorithm takes a sequence of trust observations (obtained during the evaluation period) and the HMMs corresponding to the pre-defined trust patterns as inputs and returns the category of the matching pattern. The algorithm associates the Web service to the category, which better matches the sequence, i.e., which maximizes the probability of generating the sequence.

Algorithm 1. PATTERN MATCHING ALGORITHM

inputs O : Trust observations sequence
 1: **for** each HMM_i corresponding to a trust pattern p_i **do**
 2: $Pr_i = GetProbAccept(O, HMM_i)$
 3: **end for**
 4: Compute $Pr_k = \max Pr_i$ and determine the pattern p_k
 5: **return** G_k : The category of pattern p_k

Based on the matching step, we propose a bootstrapping technique that takes into consideration two parameters: the category to which a Web service WS_i belongs and its behavior during the evaluation step. First, from the trust observations sequence O_i , an individual maliciousness rate is computed. Let O_i be an observation sequence and let $\#U_{O_i}$ and $\#T_{O_i}$ be the number of untrusted (U), respectively trusted (T), observations in O_i . The rate of maliciousness of observation O_i is defined as:

$$R_i = \frac{\#U_{O_i}}{\#U_{O_i} + \#T_{O_i}} \quad (9)$$

Intuitively, this rate indicates how many times a Web service was untrusted during the evaluation period. However, such rate is not sufficient to distinguish different Web service behaviors. For instance, it can't discriminate between a redemptive Web service who has the following behavior (e.g. *U.U.T.T*) and a betraying one (e.g. *T.T.U.U*). Distinction between those web services is possible given the category G_i each sequence is matched with.

As defined in Section 4.1, trust pattern categories exhibit different levels of trust. For instance, the level of trust in the Trustworthy category is clearly higher than the level in the Oscillating category. Similarly, the redemptive category reveals more trust than betraying category (kind of punishment). To account for these differences, a subjective category weight W_c (a value between 0 and 1) is given to each category c .

Based on the two aforementioned concepts, namely, the individual maliciousness rate and the category weight, we define the initial trust value of a Web service WS_i as follows:

Definition 2. Initial Trust Value of a Web Service

Let WS_i be a Web service and O_i a trust observations sequence representing its behavior during the evaluation period. Let G_i be the category associated to O_i according to Algorithm 1, R_i the maliciousness rate of O_i and W_{G_i} the category weight of G_i . The initial trust value of WS_i is defined as follows:

$$T_i = W_{G_i} \times (1 - R_i) \quad (10)$$

5 Experimental Analysis

The training process has been implemented using MATLAB. In addition, HMM Toolbox [10] has been installed and used to validate the results of our own implementation. As expected, both implementations (our and HMM Toolbox) returned exactly the same results. However, HMM Toolbox has been always faster due to some code optimizations. Therefore, we used HMM Toolbox in most of our experiments.

The goal of the experimental analysis is to assess the accuracy and completeness of Algorithm 1. Recall that Algorithm 1 is a classification algorithm to decide which trust category a given Web service belongs to. It is well known that each classification process is characterized by a level of accuracy, which we measure through the metrics *precision* and *recall*.

5.1 Precision and Recall

The main idea of our technique is to represent each trust pattern category using a HMM. Hence each of the 5 different trust pattern categories, namely, Trusted, Malicious, Betraying, Oscillating, and Redemptive, is associated with a different HMM. Every HMM is trained using some typical observations of the associated category. For instance, the Oscillating category HMM is trained using sequences of the form: $T.U.T.U.T.U.T.U.T.U$ and $T.T.U.U.T.T.U.U.T.U$.

After the HMMs are constructed, the experiment consists in generating a bank of trust observation sequences and see how accurate Algorithm 1 can classify them. In a nutshell, the idea is to confront the outcome of Algorithm 1 with the true category of each sequence. For the sake of conducting this experiment, the actual category of a given sequence is determined based on an expert judgement.

For each category, we define three variables:

- True Positives (tp): tracks the number of correctly classified sequences;
- False Positives (fp): tracks the number of incorrectly classified sequences;
- False Negatives (fn): tracks the number of sequences which are of the current category but classified into a different category.

These variables are updated as follows. Given an observation sequence O , Algorithm 1 is used to classify it into some category. Assume that this category is X . If the classification is correct, that is, X coincides with the actual category, the variable true positives tp for the category X is incremented. If O does not actually belong to the category X , i.e., should be classified into a different category Y , the variable false positives fp for X is incremented and the variable false negatives fn for Y is incremented.

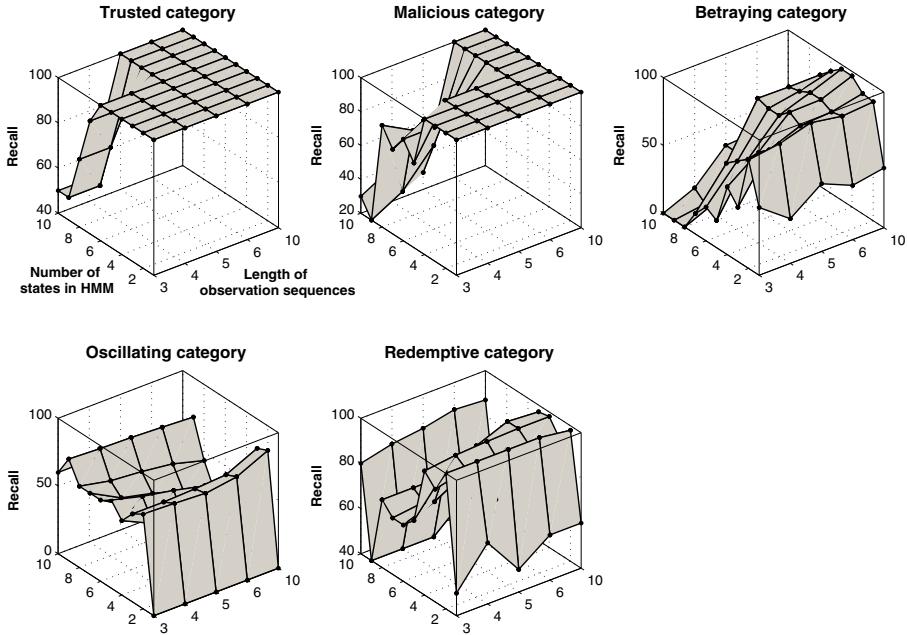


Fig. 2. The recall resulting from classifying a bank of observation sequences as we increase the number of states of the HMM (y-axis) and the length of the observation sequences (x-axis)

The accuracy of the classification for a given category can be expressed as:

$$\text{precision} = \frac{tp}{tp + fp} \quad (11)$$

On the other hand, the completeness of the algorithm for a given category is estimated by the recall metrics which is defined as:

$$\text{recall} = \frac{tp}{tp + fn} \quad (12)$$

The recall metrics estimates the percentage of sequences belonging to a certain category which has been correctly classified by the algorithm.

In order to reduce the noise factor, we repeated the experiments 10 times and we plotted the average of the 10 experiments. The results are shown in Figures 1 and 2.

Figure 1 shows, for each category, how the precision of the algorithm behaves as we increase the number of states of the HMM and the length of the observation sequences. For almost all categories, the precision increases as the length of observation sequence increases. This is normal because long sequences provide more data and consequently allow to have more accurate HMM models. As of the number of states of the HMM, Figure 1 shows that every category has an optimal number of states. For instance, Trusted and Malicious categories clearly need HMMs with small number of states (one or two). Therefore, HMMs with larger number of states will need

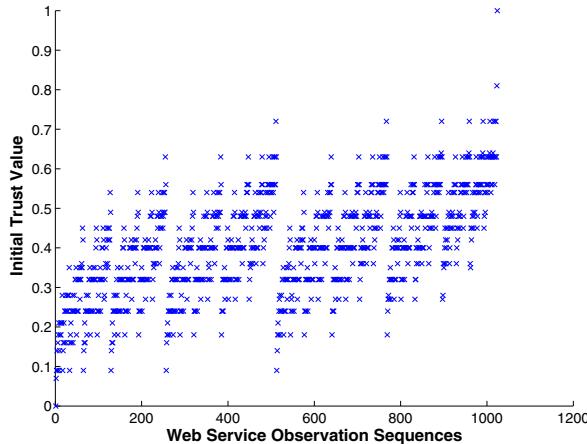


Fig. 3. Distribution of Trust Values

lengthier observation sequences to reach a precision of 100% as shown in Figure 1. For the Betraying and Redemptive categories, the optimal number of states is not very obvious. The plots suggest that the precision reaches its maximum with HMMs having four or eight states. For the Oscillating category, the plot shows clearly that the number of states of the corresponding HMM should be strictly larger than one. Overall, Figure 1 shows that for all categories except Redemptive, Algorithm 1 reaches a significantly high level of precision. For Redemptive category, this can be explained by the fact that it has a significant intersection with the Oscillating category. This is confirmed by the recall values of Oscillating category in Figure 2.

Figure 2 illustrates the recall rates for each category as we increase the number of states of the HMM and the length of observation sequences. Similarly to precision, recall increases as the observation sequences get longer. The only exception is the Oscillating category which is due to the tight intersection it has with the Redemptive category. Overall, it is easy to see that, for a particular combination of number of states in the HMM and observation sequence length, all categories reach an almost perfect recall rate of 100%.

5.2 Distribution of Trust Values

As shown earlier, the weight given to each classified trust observations sequence allows to discriminate between Web services either having the same pattern or in different patterns. In our experiments, we use the following weights: 1 for the trusted pattern, 0.6 for the malicious pattern, 0.7 for the betraying pattern, 0.8 for the oscillating pattern, and 0.9 for the redemptive pattern.

Figure 3 depicts the distribution of trust values for the classified Web services. It is worth to mention that those Web services having the same pattern are close (but still different thanks to the maliciousness rate) to each others and together they form a kind of a cluster.

6 Conclusion

We proposed in this work a new bootstrapping technique for Web services based on trust patterns and hidden markov models. The main benefits of our technique compared to the related work consist in a fair assignment of trust values to unknown Web services and a global view of possible behaviors of these services. The experimental analysis carried out showed that our HMM based classification technique is very promising and features a high rate of precision and completeness provided that the number of states of HMMs are well chosen and the observation sequences used for learning are long enough. The only concern, however, remains the difficulty of the algorithm to cope with the intersection between categories, in particular, the redemptive and oscillating categories. Another interesting research initiative is to apply our technique on real world Web services. These research directions are part of our future work.

References

1. Baum, L.E., Egon, J.A.: An Inequality with Applications to Statistical Estimation for Probabilistic Functions of a Markov Process and to a Model of Ecology. *Bull. Amer. Meteorol. Soc.* 73, 360–363 (1967)
2. Moukas, A., Zacharia, G., Maes, P.: Collaborative Reputation Mechanisms in Electronic Marketplaces. *Decision Support Systems* 29(4), 371–388 (2000)
3. Grandison, T., Sloman, M.: A survey of trust in internet applications. *IEEE Communication Surveys & Tutorials* 4(4), 2–16 (2000)
4. Malik, Z.: Reputation-based Trust Framework for Service Oriented Environments. PhD thesis, Virginia Polytechnic Institute and State University (October 2008)
5. Malik, Z., Bouguettaya, A.: RATEWeb: Reputation Assessment for Trust Establishment among Web services. *Very Large Data Bases (VLDB)* 18(4), 885–911 (2009)
6. Malik, Z., Bouguettaya, A.: Reputation Bootstrapping for Trust Establishment among Web Services. *IEEE Internet Computing* 13(1), 40–47 (2009)
7. Marti, S., Garcia-Molina, H.: Taxonomy of Trust: Categorizing P2P Reputation Systems. *Computer Networks* 50(4), 472–484 (2006)
8. Maximilien, E., Singh, M.: Reputation and Endorsement for Web Services. *SIGecom Exchanges* 3(1), 24–31 (2002)
9. Maximilien, E., Singh, M.: Toward Autonomic Web Services Trust and Selection. In: Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC 2004), New York, NY, USA, pp. 212–221 (2004)
10. Murphy, K.: Hidden Markov Model (HMM) Toolbox for Matlab (2005), <http://www.cs.ubc.ca/murphyk/Software/HMM/hmm.html>
11. Rabiner, L.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE* 77(2), 257–286 (2000)
12. Doshi, P., Paradesi, S., Swaika, S.: Integrating Behavioral Trust in Web Service Compositions. In: Proceedings of the Seventh International Conference on Web Services (ICWS 2009), Los Angeles, CA, USA, pp. 453–460 (2009)
13. OASIS Web Service Secure Exchange TC. Ws-trust 1.3 (2007), <http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html>
14. Yahyaoui, H.: Trust Assessment for Web Services. In: IEEE International Conference on Web Services (ICWS 2010), Miami, FL, USA, pp. 315–320 (2010)

Parallel Distributed Rendering of HTML5 Canvas Elements

Shohei Yokoyama and Hiroshi Ishikawa

Shizuoka University,
3-5-1 Johoku Naka-ku Hamamatsu 432-8011, Japan
yokoyama,ishikawa}@inf.shizuoka.ac.jp

Abstract. In this paper, we explain the rendering of ultra-high-resolution web content using HTML5 `<canvas>` elements. Many high-resolution massive datasets have recently been presented on the web. For example, Google Maps provides satellite images of Earth's surface and atmosphere at various resolutions. However, the scope of information that a user can view depends on the number of pixels of the user's computer monitor, irrespective of the data resolution. Therefore, we propose a parallel distributed rendering method for web content using multiple LCD monitors. We demonstrate that our system can draw an 8240 pixel \times 4920 pixel HTML5 `<canvas>` element over a 16-monitor tiled display wall.

Keywords: HTML5, Canvas Element, Parallel Rendering, Tiled Display Wall.

1 Introduction

Recently, massive datasets have become available on the web as information technology has developed. IDC reported that 998 EB of data were generated in 2010 in comparison with 161 EB in 2006 [9]. Such massive datasets are shared and exchanged over the Internet. Data use generally occurs in three stages: data generation, data processing and data visualization. Systems that handle massive datasets on the Internet must achieve high scalability at each of these three stages.

This paper describes a novel technique for realizing real-time, parallel distributed rendering of web content with user interaction. We focus on the `<canvas>` element, which is one of the newest web technologies. It is part of HTML5 and allows for dynamic scriptable rendering of 2D shapes and bitmap images. Our main contributions are summarized as follows.

- We propose a novel parallel distributed rendering technique using the HTML5 `<canvas>` element on tiled display walls.
- We demonstrate that the proposed technique achieves high scalability in rendering an ultra-high-resolution (8240 pixel \times 4920 pixel) web application.
- We also illustrate a low-cost hardware and middleware for the proposed technique of web-based parallel distributed rendering. To our knowledge, this technique has not been described previously.

A tiled display wall is a technique to build a virtual ultra-high-resolution display comprising multiple display monitors and is used to visualize ultra-high-resolution data. Many studies have investigated high-resolution displays, but most proposals targeted are for scientific and medical visualization. Although the developed techniques perform well, they have a very high cost. Furthermore, developers who work with such displays must have deep knowledge of programming and networking.

In this paper, we propose a novel technique for realizing parallel distributed rendering of the HTML5 `<canvas>` element on tiled display walls.

The population of web developers is growing at a tremendous pace. Many skilled web developers and programmers are working currently to create display applications. In addition, web browsers are used in various operating systems and apply many standards. Many web services such as Google Maps API are available on the Internet. We propose a method for using a high-resolution web application that is executed on a tiled display wall based on web standards that include web technologies such as HTML5, JavaScript and PHP.

The remainder of this paper is organized as follows. Section 2 describes related works. Section 3 explains our tiled display wall environment, which is the basis of this research. Our proposed method of realizing parallel distributed rendering of the HTML5 `<canvas>` element is described in Section 4. In Section 5, we discuss our experiments and evaluate the results. Finally, Section 6 concludes the paper.

2 Related Works

Parallel distributed rendering of high-resolution images has a long history [5]. The purpose of such research is to develop techniques for efficient rendering of three-dimensional computer graphics. OpenGL made real-time animation of computer graphics possible [2]. Subsequently, Chromium [10] unified these two technologies, the parallel rendering and the OpenGL, and realized real-time rendering of high-resolution images using a parallel distributed method.

The resolution of LCD monitor is insufficient to display high-resolution images. The best commercially available LCD monitor has WQXGA resolution (2560 pixel \times 1600 pixel). Therefore, Chromium and other technologies are designed for parallel rendering on a tiled display wall [18].

A tiled display wall is a virtual ultra-high-resolution display consisting of multiple display monitors. Figure 1 illustrates both a tiled display wall and an LCD monitor displaying the same image from Google Maps. The tiled display wall not only produces a large display but also has many pixels. Therefore, it can show high-resolution content.

Many proposed approaches are used for building tiled display wall systems. NASA's Hyperwall [15] has a 64-megapixel tiled display wall consisting of 49 monitors (7 horizontal \times 7 vertical). LambdaVision uses 55 monitors (11 horizontal \times 5 vertical) and builds a 100-megapixel high-resolution tiled display wall system. Renambot et al. who are members of LambdaVision project also



Fig. 1. Resolution of an LCD monitor and a tiled display wall

proposed middleware for tiled display walls called SAGE [14]. HIPerSpace [7], which has a 225-megapixel display, is an extremely large tiled display wall used at the University of California, San Diego. Existing tiled display wall systems were surveyed in detail by Ni et al. [13].

The performance and resolution of these systems are suited for developing applications for scientific visualization of life science data and ultra-high-resolution satellite images. Research issues for improving high-performance scientific computing related to tiled display walls have received more importance than developing consumer applications. Consequently, such applications require the use of expensive high-end machines, complex settings and extensive programming. However, web technologies are growing at a fast rate, ultra-high-resolution satellite images, e.g. Google Maps, are becoming increasingly available via web browsers and are valued by many users.

In other words, high-resolution visualization is no longer for scientists only: ordinary people can access it. Therefore, we built a low-cost tiled display wall consisting of low-end machines and based solely on web technologies. In addition, our tiled display wall system uses only web programming languages. The main purpose of this research is the rendering of <canvas> element of HTML5, which is one of the newest web technologies.

Our proposed system uses HTML5 <canvas> element and JavaScript instead of OpenGL and other graphic APIs for writing applications that produce 2D and 3D computer graphics. That is, multiple <canvas> elements that are displayed on web browsers executed on multiple computers build a virtual high-resolution and large <canvas> element. Research on distributed web interfaces has handled cooperative processing using multiple web browsers [11,19]. However, these studies do not consider whether the monitors are adjacent and do not attempt parallel rendering in the environment of the tiled display wall. Our system realizes cooperative parallel distributed rendering on a tiled display wall.

Applications of the system are implemented using common web technologies including HTML5, JavaScript and server-side scripting. Our tiled display wall middleware provides a single, extensive <canvas> area to developers although it consists of multiple <canvas> elements displayed on distributed machines. This is because the distributed environment is hidden and is realized by the middleware.

3 Web Based Tiled Display Wall

3.1 Hardware and Software Architecture

The software and hardware architecture of our web-based tiled display wall are shown in Figure 2. As shown in the figure, the system consists of multiple computers (**Receivers**), which is part of a tiled display, a computer (**Commander**) as the user interface for web applications on the tiled display wall, and a web server (**Messenger** in the figure) for synchronizing the monitors.

To develop a high-resolution web application on the tiled display wall, developers must implement two PHP programs, `commander.php` and `receiver.php`. The `commander.php` program is accessed from the **Commander**, and it includes a graphical user interface designed and implemented using HTML and JavaScript. The `receiver.php` program is accessed from the **Receiver**; it includes high-resolution

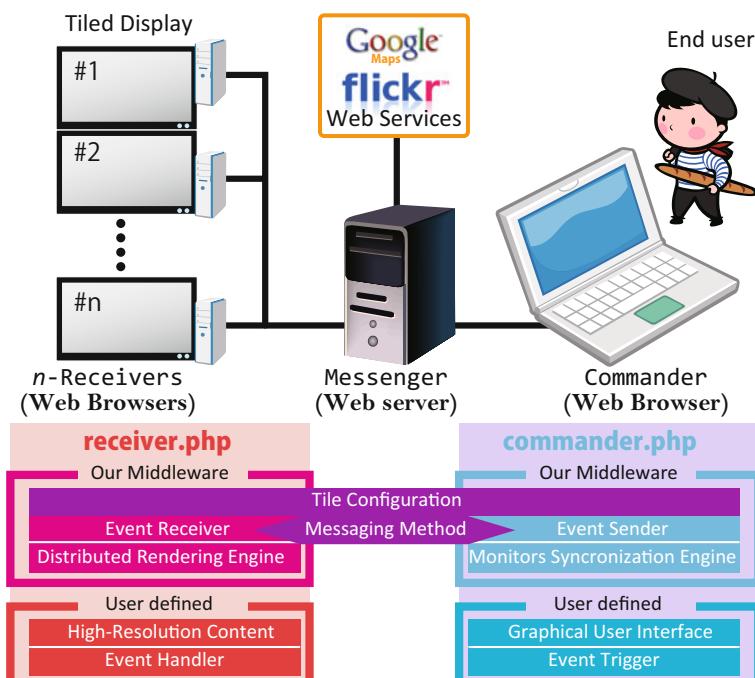


Fig. 2. Hardware and software architecture



Fig. 3. Tiled display wall

content designed using HTML and JavaScript. In addition, both PHP programs import our runtime library, which is written in JavaScript and provides a messaging method, and a configuration for the tiled display wall.

Both `commander.php` and `receiver.php` are stored on the web server (**Messenger**). Therefore, the first step in executing an application is to download `commander.php` and `receiver.php` on the **Commander** and the **Receivers**, respectively, from the **Messenger** via their web browsers.

Figure 3 shows our 16-monitor tiled display wall testbed, which uses this architecture. The tiled display wall consists of 16 full-HD (1920 pixel × 1080 pixel) LCD monitors and 16 “nettop” PCs that have Atom (Intel Corp.) processors and act as **Commanders**. Our display differs from existing tiled display walls in that the rendering engine is built on a web browser. All the **Receivers** display a web browser in a kiosk mode, a full-screen mode with no toolbars. For example, Internet Explorer can be launched in kiosk mode by using -k as a command line argument. The **Receivers** are essentially computers with simple factory settings because a web browser is pre-installed on a new computer; no extra program is necessary to build the proposed system.

The **Commander** is also a web browser. For this testbed, we select Wii (Nintendo), which has an Opera web browser (Opera Software ASA) as the **Commander**. Wii is a home video game console; an intuitive control, Wii remote, is available with it. The **Messenger** is a web server; we use Apache2 HTTP daemon on the Fedora Linux distribution (Red Hat Inc.).

3.2 Messaging

A user uses the **Commander** to control a tiled display wall application. In our testbed, the user holds a Wii remote. As the rendering is done on the **Receivers**, the user’s interactions must be sent from the **Commander** to all the **Receivers**. Our middleware’s messaging method is used for this purpose. Event triggers can be defined in `commander.php` and the event handler can be defined in `receiver.php`.

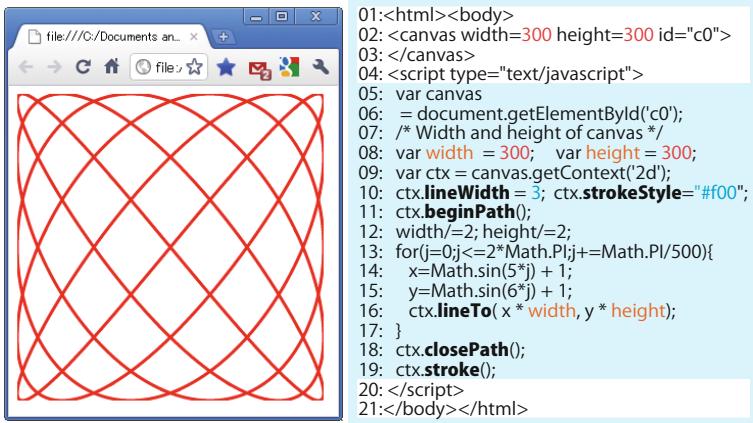


Fig. 4. JavaScript code for rendering a Lissajous curve

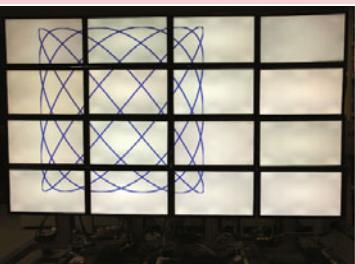
The event consists of destinations, an identifier and options (arguments). Our middleware sends the message from the **Commander** to the **Receivers** specified as the destination. Our tiled display wall system is described in detail in our previous work [3,20].

4 Distributed Parallel Rendering

4.1 HTML5 <canvas>

The `<canvas>` element is part of HTML5 and allows for dynamic, scriptable rendering of 2D shapes and bitmap images. It is the first graphic API for pure HTML content, and it is expected to increase the expressiveness of web content. Figure 4 shows the JavaScript code for drawing a Lissajous curve and a screenshot of a web browser showing the Lissajous curve rendered on the `<canvas>` element. In the figure, the drawing line is defined between the `beginPath` method and the `stroke` method. The filled shape is rendered using a `fill` method instead of the `stroke` method. Other drawing methods include the `bezierCurveTo` method for drawing a bezier curve, the `arcTo` method for drawing an arc and the `drawImage` method for displaying image files such as .jpeg, .png and .gif. The colour and width of the line are varied by changing the properties of the context object (line 10).

We apply the `<canvas>` element and graphic API to parallel distributed rendering on tiled display walls. A `<canvas>` element generally cannot be placed over an entire tiled display wall because the wall is a distributed system consisting of more than one computer. However, our proposed method virtualizes multiple `<canvas>` elements as a single huge high-resolution `<canvas>` element over an entire tiled display wall. This method is described in the next section.



```

receiver.php
01:<html><head>
02:<?php echo $WallDisplay_LIBLARY; ?>
03:<script type="text/javascript">
04: Ext.onReady(function(){
05:   receiver
06:   = new Wdm.Receiver(tileConfigure);
07:   receiver.start();
08:});</script>
09:</head></html>

commander.php
01:<html><head>
02:<?php echo $WallDisplay_LIBLARY; ?>
03:<script type="text/javascript">
04: Ext.onReady(function(){
05:   var commander
06:   = new Wdm.Commander(tileConfigure);
07:   commander.start();
08:   /* Width and height of canvas */
09:   var width = 4000; var height = 4000;
10:   var canvas = commander.createCanvas(
11:     {top: 300, left: 900,
12:      height:height,width:width},px");
13:   var ctx = canvas.getContext('2d');
14:   ctx.lineWidth = 30; ctx.strokeStyle = "#00f";
15:   ctx.beginPath();
16:   width/=2; height/=2; var xy;
17:   for(j=0;j<=2*Math.PI;j+=Math.PI/500){
18:     x=Math.sin(5*j) + 1;
19:     y=Math.sin(6*j) + 1;
20:     ctx.lineTo(x*width,y*height);
21:   }
22:   ctx.closePath();
23:   ctx.stroke();
24:});</script>
25:</head></html>

```

Fig. 5. JavaScript code for distributed rendering of a Lissajous curve

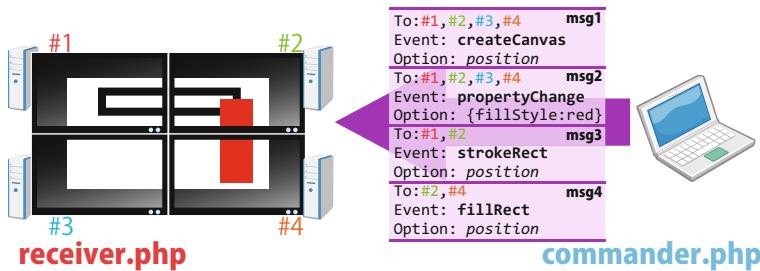


Fig. 6. Messages for drawing two rectangles on a tiled display wall

4.2 Rendering from commander.php

The code for distributed rendering of a `<canvas>` element is written in either `commander.php` or `receiver.php`. First, we describe the method of rendering from `commander.php`.

Figure 5 is an example of code that draws a 4000 pixel \times 4000 pixel Lissajous curve on a tiled display wall. As shown here, the code that renders shapes and images on the tiled display wall is the same as that shown in Figure 4. However, the `Canvas` object (line 10) and the `Context` object (line 13) are our wrappers for a native object of the `<canvas>` element. If a method, e.g. `lineTo` (line 20), of the `Context` wrapper is called, our middleware sends the event to the `Receivers` via the messaging mechanism.

Although we use a complex system consisting of many computers, a developer can write code for a 4000 pixel \times 4000 pixel `<canvas>` element without any knowledge of networking or a distributed environment. Our middleware delivers code appropriately to the corresponding `Receivers` on the tiled monitors.

```

01:<html>
02:<?php echo $JAVASCRIPT_LIBRARY; ?>
03:<script type="text/javascript">
04: Ext.onReady(function(){
05:   receiver = new Wdm.Receiver(configure);
06:   var LissajousDrawer = Ext.extend(Wdm.Drawer,{
07:     constructor:function() {
08:       LissajousDrawer.superclass.constructor.apply(this, arguments);
09:       this.c=5;
10:     },
11:     draw:function(ctx){
12:       var width = this.width;
13:       var height = this.height;
14:       ctx.lineWidth = 30;
15:       ctx.strokeStyle = "#00f";
16:       ctx.beginPath();
17:       width/=2; height/=2; var x,y;
18:       for(j=0;j<=2*Math.PI;j+=Math.PI/500){
19:         x=Math.sin(((this.c++)%15)*j) + 1;
20:         y=Math.sin(((this.c++)%15)*j) + 1;
21:         ctx.lineTo(x*width,y*height);
22:       }
23:       ctx.closePath();
24:       ctx.stroke();
25:     },
26:     runner_mode:"sleep",
27:     runner_timing:1000
28:   };//draw() is called every 1000 msec*/
29: receiver.setDrawer("CURVE",LissajousDrawer);
30: receiver.start();
31: });
32:</script>
33:</head></html>

```

receiver.php

```

01:<html><head>
02:<?= $JAVASCRIPT_LIBRARY ?>
03:<script type="text/javascript">
04: Ext.onReady(function(){
05:   var commander
06:     = new Wdm.Commander(configure);
07:   commander.start();
08:   var canvas = commander.createCanvas(
09:     {"top": .300, "left": .900,
10:      "height":4000,"width":4000}, "px",
11:      "CURVE");
12:   });
13:</script>
14:</head></html>

```

commander.php

Fig. 7. JavaScript code for parallel distributed rendering of a Lissajous curve

Figure 6 shows examples of messages that are sent and received between the **Commander** and the **Receivers**. These messages create a `<canvas>` element and draw two rectangles, one black and one red. The black rectangle is drawn on monitors #1 and #2; therefore, the message is sent to the #1 and #2 `receiver.php` files. The red rectangle is drawn on monitors #2 and #4; therefore, the message is sent to the #2 and #4 `receiver.php` files. Thus, only monitors that must display part of a shape receive the message containing the draw event. This mechanism reduces network traffic between the **Commander** and the **Receivers**.

Because `commander.php` also shows a graphical user interface and handles event listeners that respond to user actions, a dynamic drawing based on user actions is suitable for this method. This method is a distributed rendering but not a parallel rendering because the drawing commands are centralized in `commander.php`. In addition, the latency between the **Commander** and the **Receivers** cannot be bypassed, particularly for animations. For this purpose, the rendering command is written in `receiver.php`. The screen shown in Figure 6 is drawn using these methods.

4.3 Rendering from `receiver.php`

The drawing commands written in `receiver.php` are delivered to all **Receivers** and executed in parallel. Figure 7 shows an example code for drawing different

Lissajous curves every second. The `receiver.php` file is downloaded by all the `Receivers`, but drawing commands that do not include the area of the relevant monitor are ignored.

The `LissajousDrawer` class, which extends `Wdm.Drawer`, is defined in `receiver.php` in Figure 7. `LissajousDrawer` has a constructor, a draw method and some properties. The draw method is called constantly, and the runner timing property (line 26 in `receiver.php`) is the sleep time between the method calls. The size and position of the `<canvas>` element are defined in `commander.php` (lines 09,10). The `createCanvas` method's third argument, CURVE, is an identifier of a `Drawer` instance. The identifier also appears in the argument of the `setDrawer` method (line 29 in `commander.php`). The `Canvas` object in `commander.php` and the `Drawer` object in `receiver.php` are bound together by the identifier.

5 Performance Evaluation

5.1 Photomosaic

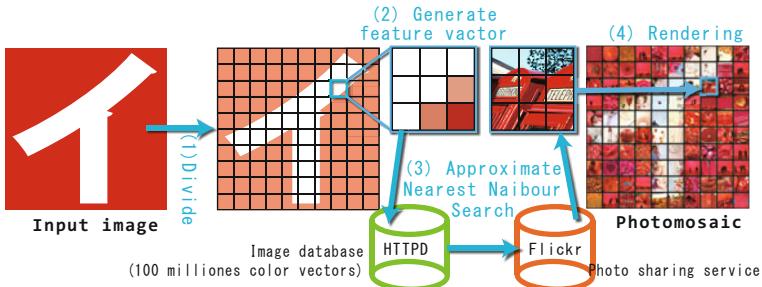
To test our method, we implemented a complex rendering application consisting of a high-resolution photomosaic renderer and measured the parallel distributed rendering performance.

The photomosaic technique transforms an input image into a rectangular grid of small images. One of the earliest concepts of the photomosaic was presented by Salvador Dali [6]. His painting “Gala Contemplating the Mediterranean Sea which at Twenty Meters becomes a Portrait of Abraham Lincoln (Homage to Rothko)” looks like a portrait of Lincoln, but a nude of his wife Gala and a small portrait of Lincoln actually appear in the painting. This is the best-known image made from many other images. Robert Silvers, a student at MIT Media Lab, proposed a computer-aided photomosaic [16]. One of his masterpieces is a portrait of Mickey Mouse composed from many scenes from Walt Disney’s movies [17].

The algorithm for generating a computer-aided photomosaic is as follows:

1. Divide an input image into small rectangular areas.
2. Search for images similar to each area from an image database.
3. Replace the areas with the similar images.

That is, the principal part of the algorithm consists of repeated searches for similar images. Blasi et al. proposed an efficient photomosaic generation method based on approximate nearest neighbour searching Blasi et al. [4,8] They showed that a 1024 pixel \times 768 pixel photomosaic is created in about 32 s using a database of 1417 images of 10 pixel \times 10 pixel on a personal computer that has an Athlon XP-M 1800+ CPU and 192 MB of RAM. This speed is sufficient to be used for a personal computer monitor, but the resolution of computer monitors is insufficient to render photomosaics. A high-quality photomosaic requires a high-resolution canvas; consequently, the photomosaic technique is often used for printed matter. However, by using the proposed method, a high-quality photomosaic having resolution sufficient for printing can be rendered dynamically

**Fig. 8.** Photomosaic generation

on a high-resolution tiled display wall. This is why we selected photomosaic generation as a benchmark of the proposed system.

The other important problem of photomosaic generation is the database size of the small images. The quality of a photomosaic is influenced not only by the number of pixels but also by the size of the database. This problem is solved by using the web because many photo-sharing websites share billions of images. In this section, we measure the performance of the entire system and assess the effectiveness of the proposed method using photomosaic generation based on photos on Flickr [1].

5.2 Setting

The benchmark application creates photomosaic images using large image sets from Flickr, a public image-hosting service. We crawled pictures from Flickr, divided each of them into nine sections and extracted the average colour of each section. We used one million colour vectors for this experiment and stored them in a database. A colour vector consists of red, green and blue levels, saturation and brightness of the nine subsections; therefore, one small image has a 45-dimensional vector. The specifications of the machines are listed in Table 1.

Figure 8 illustrates the photomosaic generation algorithm. The input photograph is subdivided, and each subsection is compared with the colour vectors of the image database. This process yields a Flickr URL for the closest match. That is, a list of URLs is generated for a given image, the images are downloaded from Flickr, and the photomosaic is drawn on a tiled display wall. Every **Receiver** searches and draws in parallel (see Figure 11 in the Appendix).

Table 1. Machine specifications

Messenger		Receiver	
M/B	Asus P6T7 WS SuperComputer	Model	Acer Aspire Revo
CPU	Intel Core i7 975EE (Quad Core)	CPU	Intel Atom Processor
Memory	12GB	Memory	2GB
OS	Linux (Fedora12)	OS	Windows Vista

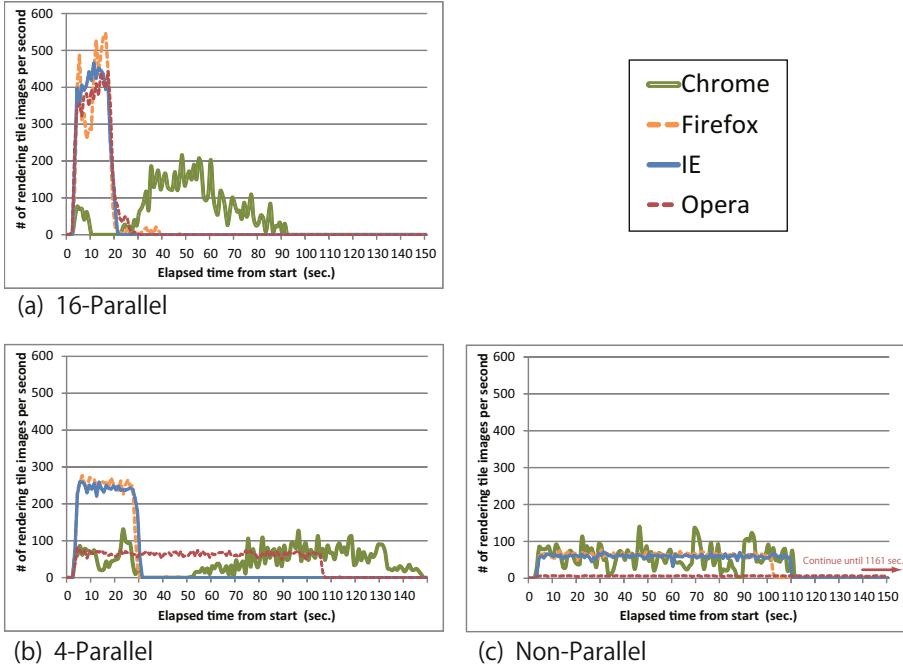


Fig. 9. Throughput of parallel distributed `<canvas>` rendering

Our testbed has a 8240 pixel \times 4920 pixel display consisting of 16 monitors. We used one million Flickr thumbnail images of 75 pixel \times 75 pixel resolution and measured the execution time to create and draw an 8240 pixel \times 4920 pixel photomosaic using 16 monitors, four monitors and one monitor. When four monitors and one monitor were used, the photomosaic was drawn into a scrollable area because it was larger than the screen. We also used the ANN library [12] to search for similar images. The web browser on the Receivers is Chrome 9.0, Firefox 3.6.13, Opera 11.01 or Internet Explore 9 Beta. The results are the mean values of five executions.

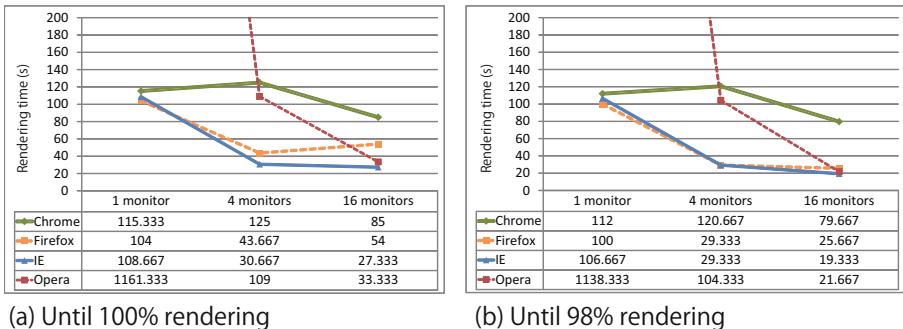
5.3 Result

Photomosaic rendering requires approximately 650 nearest neighbour searches in a short period. A few time-outs occasionally occur when 6500 thumbnail images are downloaded from Flickr. In this case, the application can send a re-send request to Flickr, but we ignore the time-outs in this experiment because the purpose is to estimate the performance of the proposed system, not error recovery. Table 2 shows the mean number of error images and thumbnail images downloaded from Flickr.

The number of thumbnails of each condition take different values. This is because when nearest neighbour searches occasionally return the same thumbnails, the web browser loads the image from cache memory.

Table 2. Average number of drawing tile images and HTTP errors

	Chrome		Firefox		IE		Opera	
	errors	images	errors	images	errors	images	errors	images
1 monitor	0.0	6350.0	2.7	6324.0	1.7	6346.0	1.3	7303.7
4 monitors	0.7	6457.0	3.3	6329.0	1.7	6352.7	1.0	6657.7
16 monitors	2.3	6620.3	25.0	6401.3	3.7	6515.7	1.0	6498.7

**Fig. 10.** Rendering time

The results of this experiment are presented in Figure 9. Figure 9(a) uses all 16 monitors of the tiled display wall. All web browsers except Google Chrome drew more than 400 thumbnail images per second at their peak performance. In the case of four monitors, the peak performance of Firefox and Internet Explorer was approximately 250 images per second. In the case of one monitor (non-parallel conditions), the peak performance of Firefox and Internet Explorer was about 80 images per second. These results suggest that the throughput of parallel distributed <canvas> rendering is proportional to the degree of parallelism.

The rendering time is presented in Figure 10(a). The result includes an unexpected delay in HTTP response from Flickr. Therefore, we also measured the time to draw 98% of all the thumbnails [Figure 10(b)]. The fastest case was 27 s to create a photomosaic in Internet Explorer. In Firefox, the photomosaic was created in 54 s, but the time required to draw 98% of the thumbnails shown in Figure 10(b) is 25 s, which is similar to the other times. Opera shows the same trend, but the result of nonparallel operation is worse than the others. Only Chrome shows a different trend; it is slowest in the case of four parallel monitors. We think the reason is that Chrome has a high speed of operation. This is because a web browser on a **Receiver** sends many HTTP requests for the nearest neighbour search on the **Messenger**, exceeding its capacity and occupying all the reserved ports. Consequently, the other **Receivers** wait for completion of the drawing by the **Receiver**.

Finally, we present the memory consumption of Internet Explorer and Firefox in Table 3. The values represent the difference between the memory used before and after photomosaic rendering.

Table 3. Memory consumption

	Firefox	IE
1 monitor	67.8	232.7
4 monitors	35.9	41.8
16 monitors	17.9	8.4

(MB)

Memory consumption might be proportional to the size of the <canvas> element. This is because the memory consumption of 16 parallel monitors is less than that of the others. The result shows this trend. These experiments clarify that a web-based tiled display wall achieves effective parallel distributed rendering of <canvas> elements.

6 Conclusion

As described in this paper, we propose a method of parallel distributed rendering of the HTML5 <canvas> element. We also describe the design of a web-based tiled display wall system. The experimental results show that the proposed system is highly efficient and scalable. High-resolution web content realized by this research will bring about qualitative changes in Internet applications.

This work has inspired several ideas:

- We implements WebSocket messaging between the **Commander** and the **Receivers**.
- We will attempt to improve the latency of distributed frame synchronisation and the timing of the draw method call of the Drawer object in receiver.php.

References

1. Flickr from yahoo, <http://flickr.com/>
2. OpenGL, <http://www.opengl.org/>
3. Wall display in mosaic, http://shohei.yokoyama.ac/Wall_Display_in_Mosaic/en
4. Blasi, G.D., Petralia, M.: Fast Photomosaic. In: Poster Proceedings of ACM/WSCG 2005, Citeseer (2005)
5. Crockett, T.: An Introduction to Parallel Rendering. Parallel Computing 23(7), 819–843 (1997)
6. Dali, S.: Gala contemplating the Mediterranean Sea, which at twenty meters becomes the portrait of Abraham Lincoln (1976)
7. DeFanti, T.A., Leigh, J., Renambot, L., Jeong, B., Verlo, A., Long, L., Brown, M., Sandin, D.J., Vishwanath, V., Liu, Q., Katz, M.J., Papadopoulos, P., Keefe, J.P., Hidley, G.R., Dawe, G.L., Kaufman, I., Glogowski, B., Doerr, K.-U., Singh, R., Girado, J., Schulze, J.P., Kuester, F., Smarr, L.: The optiportal, a scalable visualization, storage, and computing interface device for the optiputer. Future Generation Computer Systems, The International Journal of Grid Computing and eScience 25(2), 114–123 (2009)

8. di Blasi, G., Gallo, G., Petrali, M.P.: Smart ideas for photomosaic rendering. In: Eurographics Italian Chapter Conference 2006, pp. 267–272 (2006)
9. Fantz, J.F., Reinsel, D., Chute, C., Schlichting, W., McArthur, J., Minton, S., Xheneti, I., Tonoheva, A., Manfrediz, A.: The expanding digital universe. In: An IDC White Paper - Sponsored by EMC (2007)
10. Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P.D., Klosowski, J.T.: Chromium: A stream-processing framework for interactive rendering on clusters. In: ACM SIGGRAPH 2002, pp. 693–702 (2002)
11. Melchior, J., Grolaux, D., Vanderdonckt, J., Roy, P.V.: A toolkit for peer-to-peer distributed user interfaces: Concepts, implementation, and applications. In: Proceedings of The ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2009), pp. 69–78 (2009)
12. Mount, D.M., Arya, S.: Ann: A library for approximate nearest neighbor searching, <http://www.cs.umd.edu/~mount/ANN/>
13. Ni, T., Schmidt, G.S., Staadt, O.G., Livingston, M.A., Ball, R., May, R.: A survey on large high-resolution display technologies, techniques, and applications. In: Virtual Reality Conference, 2006, pp. 223–236 (2006)
14. Renambot, L., Rao, A., Singh, R., Byungil, J., Krishnaprasad, N., Vishwanath, V., Vaidya, C., Nicholas, S., Spale, A., Charles, Z., Gideon, G., Leigh, J., Johnson, A.: Sage: the scalable adaptive graphics environment. In: Workshop on Advanced Collaborative Environments, WACE 2004 (2004)
15. Sandstrom, T.A., Henze, C., Levit, C.: The hyperwall. In: Proceedings of International Conference on Coordinated and Multiple Views in Exploratory Visualization, pp. 124–133 (2003)
16. Silvers, R.: Photomosaics: Putting Pictures in Their Place. PhD thesis, Massachusetts Institute of Technology (1996)
17. Silvers, R., Tieman, R.: Disney's Photomosaics. Hyperion (1998)
18. Staadt, O., Walker, J., Nuber, C., Hamann, B.: A Survey and Performance Analysis of Software Platforms for Interactive Cluster-based Multi-screen Rendering. In: ACM SIGGRAPH ASIA 2008 Courses, pp. 1–10. ACM, New York (2008)
19. Vandervelpen, C., Vanderhulst, G., Luyten, K., Coninx, K.: Light-weight distributed web interfaces: Preparing the web for heterogeneous environments. In: Lowe, D.G., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 197–202. Springer, Heidelberg (2005)
20. Yokoyama, S., Ishikawa, H.: Creating Decomposable Web Applications On High-Resolution Tiled Display Walls. In: Proceedings of the IADIS International Conference on WWW/Internet, pp. 151–158 (2010)

Appendix: Parallel Distributed Photomosaic Rendering



Fig. 11. Parallel distributed photomosaic rendering in 20 s. A movie version is available to the public at <http://www.youtube.com/watch?v=ECZ03giPIXE>.

Formal Modeling of RESTful Systems Using Finite-State Machines

Ivan Zuzak, Ivan Budiselic, and Goran Delac

School of Electrical Engineering and Computing, University of Zagreb,

Unska 3, 10000 Zagreb, Croatia

{izuzak,ibudiselic,gdelach}@gmail.com

Abstract. Representational State Transfer (REST), as an architectural style for distributed hypermedia systems, enables scalable operation of the World Wide Web (WWW) and is the foundation for its future evolution. However, although described over 10 years ago, no comprehensive formal model for representing RESTful systems exists. The lack of a formal model has hindered understanding of the REST architectural style and the WWW architecture, consequently limiting Web engineering advancement. In this paper we present a model of RESTful systems based on a finite-state machine formalism. We show that the model enables intuitive formalization of many REST's constraints, including uniform interface, stateless client-server operation, and code-on-demand execution. We describe the model's mapping to a system-level view of operation and apply the model to an example Web application. Finally, we outline benefits of the model, ranging from better understanding of REST to designing frameworks for RESTful system development.

Keywords: representational state transfer, World Wide Web, software architectural styles, formal model, finite-state machines, hypermedia.

1 Introduction

One of the main reasons for the wide adoption of the World Wide Web (WWW) as a global information system has been its ability to scale and remain reliable with the rapid growth in the number of its users and applications. Enabling this growth is an architecture [1] designed just for the purpose of developing large-scale distributed hypermedia systems such as the WWW. The foundation of this architecture is a set of software design principles named the *Representational State Transfer* architectural style (REST) [2]. In a way, REST describes how a Web application should behave in order to maximize beneficial properties, such as simplicity, evolvability, and performance.

From its introduction in year 2000., REST has not only guided many incremental changes in WWW's continuous evolution, such as the recently standardized HTTP PATCH method [3], but has also been guiding the development of its new dimensions in order to preserve its desirable properties. These efforts include the expansion of the WWW with higher-level applications, interlinked

data, physical devices and real-time access, through mashups [7], the Semantic Web [5], Web of Things [4] and the Real-Time Web [6]. However, as the WWW grows in functionality, it also grows in complexity and is consequently becoming harder to understand and explain at the architectural level. Therefore, understanding REST is essential for engineering the WWW and its future.

However, although defined over 10 years ago, REST's architectural principles have only been semi-formally described using diagrams, tabular techniques and natural language descriptions. Furthermore, although formal models of hypermedia systems in general do exist [9], no such model covers fundamental principles of REST and most techniques are used to model the WWW which includes many unRESTful properties. In result, no formalism for modeling RESTful systems exists today. This lack of formal explanation has increasingly been causing negative effects, such as misunderstanding of REST concepts, misuse of terminology [10] and ignorance of benefits of the REST style. For example, common misunderstandings include the overload in meaning of the word state, such as state, application state, resource state and session state [13], and identifying functionality of REST user agents with Web browsers [14].

In result, WWW researchers and engineers experience difficulty in concisely explaining both small-scale and large-scale WWW patterns or requirements, such as defining Web application interaction [11] and defining future WWW architectural goals [12]. Furthermore, development of systems which adhere to the REST style is difficult due to a lack of software frameworks which guide their implementation [15]. This is especially true for developing machine-driven RESTful clients and their application in machine-to-machine RESTful interaction and service composition [8]. We believe this to be a direct consequence of the absence of formal models which are used as the practical encoding of general architectural principles and serve as the foundation for the software development process in such frameworks.

In this paper we present a finite-state machine (FSM) [16] formalism for modeling RESTful systems, with the primary motivation of contributing to the understanding of the REST style. Our choice of using a FSM formalism was inspired by *The Rule of Least Power* [17] which originally suggests that the least powerful language suitable for expressing constraints or solving a problem should be chosen. Consequently, one of our goals was to explore the possible limits of the FSM formalism for this specific purpose in order to suggest the use of more a powerful model. Furthermore, one of the core principles of the REST style, that resource representation transfers are used for transitioning agents from one state to another, suggest the usage of a state transition system formalism.

Our model is based on the nondeterministic finite-state machine formalism with epsilon transitions (ε -NFA). We first explain the mapping of the model's abstract elements to those of a RESTful system. In order to illustrate model usage, we introduce an example Web application and present its ε -NFA model. Next, we explain how each of REST's style constraints map to the model, including uniform interface, stateless client-server operation, and code-on-demand execution. We show that the transition function of the ε -NFA enables formalization

of the transformation of the system's application state, following the *hypermedia as the engine of application state* principle. Furthermore, we show that nondeterministic transitions of the model enable formalization of the temporally-varying mapping of resources to representations and that ε -transitions enable formalization of code-on-demand execution. The presented model naturally translates to the client-centric view of RESTful system operation with the client storing application state, issuing resource manipulation requests and integrating responses into application state, while the server performs request processing.

The remainder of the paper is organized as follows. In Section 2, we give an overview of related work, focusing on approaches for formalization of RESTful systems, hypermedia systems and Web applications. Section 3 defines our finite-state machine formalism for modeling RESTful systems and presents the model of an example Web application. In Section 4 we conclude the paper, discuss the limitations of the presented model and give directions for future work.

2 Related Work

This Section gives an overview of existing approaches for modeling RESTful systems with the goal of examining the degree of completeness in which REST principles are covered by each model. We first give an overview of related work focused on REST and similar styles and then of related work focused on modeling Web applications and hypermedia systems in general. In summary, our analysis shows several issues with existing formal and semi-formal models that motivate our research. First, most models are not focused on REST, rather on hypermedia applications in general or Web applications, and thus do not include many of REST's principles. Second, most models do not offer an explicit mapping from REST's principles to the chosen formalism, and in general do not use the terminology originally proposed for REST. Third, most models address only REST's static properties or do not offer a mapping of REST principles to a system-level view of operation dynamics. Fourth, some principles of REST are rarely included in models, such as the temporally-varying mapping of resources to representations, code-on-demand execution and steady application states.

In [18], the authors present Alfa, a framework for characterization of architectural styles, based on composing a small set of architectural primitives. The authors use Alfa to describe many architectural styles, including a subset of REST, the layered-client-code-on-demand-cache-stateless-server (LCCOD\$SS) style. However, this style does not include the uniform interface constraint, one of the most important and distinct principles of REST, while its model does not explain key REST concepts, such as resources, representations and media types.

In [19] the authors present a definition of RESTful semantic Web Services using a process calculus formalism. In this model, a RESTful system is described as a set of processes, representing origin servers and user agents, which exchange request and response messages over uniquely identified channels. This approach is very promising as it allows that channel names exchanged between processes be used to model the exchange of messages containing resource identifiers. This

property of the model enables formalization of the REST constraint of using hypermedia links as the engine of application state. We encourage further work on using process algebras for modeling RESTful systems which would include a mapping of resources, representations, media types, steady and transition states to such a model together with a generalization of the model which would not be bound to standard HTTP methods as it currently is.

In [20] the authors present a promising formal model for specifying RESTful execution of processes specified by Service nets, a specific class of Petri nets that include value passing. The main advantage of the model is its integration of hypermedia-driven application flow while its main use is in modeling composition of RESTful processes. However, the model is not explicit on where application state is stored and does not explain its transformation in response to initial fetches of resource representations which occur at the beginning of an application flow. Furthermore, the model introduces a notion of static and dynamic ports, metaphors for static and temporary resources, which is not RESTful since clients never know and do not need to know if a resource is static or temporary.

In [21] the authors present the Resource Linking Language, an XML-based language for describing interlinked REST resources and consequently the service that can be accessed by interacting with those resources. The language is based on a RESTful service description metamodel, formalized as a UML class diagram, and which incorporates many REST's concepts, such as resources, representations, media types and links. However, the static metamodel does not explicitly express the important dynamic properties of REST, such as the application state contained on the client side and the effects of the code-on-demand constraint, and does not map REST's concepts to a client-server architecture.

In [22], an agent-based model of RESTful applications is presented. In the model, an agent represents the client side of the application while the environment represents the server side. The agent has several pools of predefined logic, including application, action and protocol logic, formalized as a hierarchical state machine which drives the agent's action selection. Although explaining high-level dynamics of a RESTful system, the model is more descriptive than formal, not providing an explicit mapping of many REST's principles to the model, including code-on-demand execution, resource representations and temporally-varying mapping of resources to representations.

In [23], the authors present a broad overview of modeling methods for Web application verification and testing, using a categorization of criteria for classifying models of Web application. Although some criteria may be regarded as reflections of REST's principles, such as the *dynamic navigation* criterion which asserts the possibility of modeling servers that may nondeterministically return responses for the same requests, this work is focused on Web applications only and most principles of REST were not considered. For example, in [25], the authors introduce a finite-state machine behavioral modeling approach for hypermedia Web applications. The model is based on presenting Web pages as states and links as transitions in a FSM. Furthermore, the authors define multiple types of pages and transitions in order to model activity-initiated transitions and

automatic transitions. However, the model is based on a deterministic FSM and does not explain the temporally-varying mapping of resources to representations in RESTful systems. Furthermore, the model is based on using only “clickable links” for transitions, i.e. only navigation is used for changing application state, which is an incomplete definition of transitions in RESTful systems.

In [9], the authors give a broad systematization of formal and semi-formal reference models for hypermedia systems and a comparison of hypermedia engineering methodologies. Hypermedia reference models capture important abstractions found in hypermedia applications and describe the basic concepts of these systems, such as the node/link structure. Semi-formal models include the Amsterdam Hypermedia model while formal models include the Trellis and Dexter reference models. However, although these models describe the mechanisms by which the links and nodes in the hypermedia network are related, these do not include many principles, concepts and terminology of RESTful systems. For example, the Dexter reference model uses components and instances, while REST uses resources, representations and application state. Furthermore, these models do not offer a dynamic operational system-level view which maps system components to clients, servers and intermediaries.

In [24] the authors present an automata-based model of hyperdocuments with the goal of verifying trace-based properties by model checking. The model is focused on simple hyperlink-based connectedness of hypertext documents for the Trellis hypermedia system, which does not include important properties of RESTful systems, such as the uniform interface constraint. However, two interesting ideas are presented. First, the underlying model upon which a link automaton is constructed is based on place/transition nets in order to allow modeling of parallel execution of hyperdocuments. Second, the authors present a temporal logic for model checking link automata.

3 A Finite-State Machine Model of RESTful Systems

Finite-state machines (FSMs) are a mathematical formalism for describing processes with a finite number of possible states and sequential state transitions. Although components of a RESTful system may be viewed as separate agents, each driven by a self-contained FSM, we model the operation of the complete system, often called an *application*, as a single FSM. For formalizing RESTful systems, we use a nondeterministic finite-state machine with ε -transitions (ε -NFA), an extension of the basic deterministic FSM model. Our model is focused more on explaining the operation of RESTful systems and less on explaining their static properties. The central part of this view is the application state of a RESTful system, its definition, transformation and relation to other concepts.

In the following subsections we first give an overview of the model, formalizing elements of the ε -NFA in context of RESTful systems. Second, in order to illustrate the usage of the model, we introduce an example Web application and present its ε -NFA formalization. Next, we describe the model in more detail by mapping style constraints of REST to the presented model, including

client-server, stateless, code on demand and uniform interface styles. The presented model does not explicitly formalize the layered and cacheable constraints of REST since these are not essential for understanding the operation of a system from a functional perspective.

3.1 Model Overview

A nondeterministic finite-state machine with epsilon transitions (ε -NFA) is a tuple $(S, \Sigma, s_0, \delta, F)$ where S is a finite, non-empty set of states, Σ is a finite, non-empty set of symbols representing the input alphabet, $s_0 \in S$ is the initial state of the ε -NFA, δ is the state transition function $\delta : S \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(S)$, where $\mathcal{P}(S)$ is the power set of S , and $F \subseteq S$ is the set of accepting states. A system-level perspective of ε -NFA operation is shown in Fig. 1. The *Input Symbol Generator* module generates an input symbol based on internal rules or environment state (1). Since the generator is not part of the ε -NFA's formal model but is required to properly model its operation, we define that the generator has access to the system's state stored in the *Current State* module. The *Transition Function* accepts the generated input symbol and the current state (2), determines the next state and stores it in the *Current State* module (3). The described cycle is then repeated. Since the ε -NFA is nondeterministic, formally the Transition Function module returns a set of states, for which the practical meaning is that the system may be in any single state from that set. If at some point at least one state stored in the *Current State* module is marked as accepting, it is said that the ε -NFA accepts the sequence of input symbols read up to that point. Furthermore, since the ε -NFA includes ε -transitions, the Transition Function does not need to read an input symbol in order to perform some transitions.

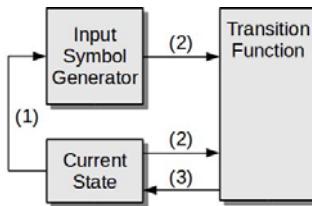


Fig. 1. System-level view of ε -NFA operation

We map a RESTful system [2] to the ε -NFA formal model as follows. Let *ResIDs* be a finite set of identifiers of system resources, let *Metas* be a finite set of metadata key-value pairs, let *LTypes* be a finite set of link types and let *LRelets* be a finite set of link relations. Furthermore, let *Links* be the finite set of hypermedia links, each defined by a resource identifier, link type and link relation $Links \subseteq ResIDs \times LTypes \times LRelets$ and let *MTypes* be a finite set of media types which determine the set of hypermedia links present in a representation $MTypes = \{MType : Reprs \rightarrow \mathcal{P}(Links)\}$. Finally, let *Reprs* be the finite

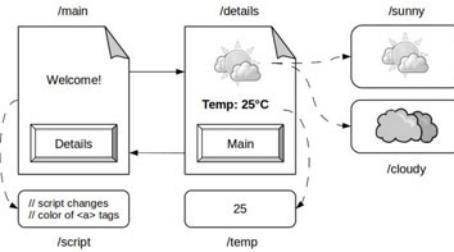
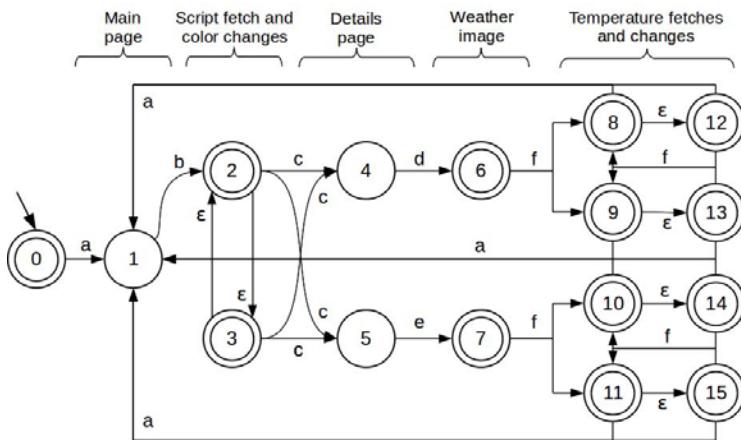
set of resource representations, each consisting of resource metadata and data $Reprs \subseteq data \times \mathcal{P}(Metas)$ where one metadata element defines the media type of the representation. The **set of states S of the ε -NFA** represents the application states of the system, $S = AppStates$, where an application state is defined as a non-empty, ordered set of representations, $AppStates \subseteq \mathcal{P}(Reprs) - \{\}$. Furthermore, the **initial state s_0 of the ε -NFA** represents the initial application state at system startup. Finally, let $Steadys$ be the subset of application states, $Steadys \subseteq AppStates$, for which it holds true that representations of all embeddable resources linked to from the first representation in the application state are also present in the application state. The **set of accepting states F of the ε -NFA** represents the steady application states, $F = Steadys$.

Next, let Ops be a finite set of resource manipulation methods, and let $Reqs$ be the finite set of valid resource manipulation requests $Reqs \subseteq Ops \times ResIDs \times Reprs$. The **set of input symbols Σ of the ε -NFA** represents requests $Reqs$ and their corresponding link types $LTypes$ for manipulating resources, $\Sigma \subseteq Reqs \times LTypes$. Furthermore, let $Resrcs$ be a finite set of resources, mappings from a resource identifier to a representation $Resrcs : \{Resrc : ResIDs \rightarrow Reprs\}$, and $Resps$ be a finite set of resource manipulation responses $Resps \subseteq ResIDs \times Reprs$. The **transition function δ of the ε -NFA** represents the translation of input symbols into requests, processing of requests into responses and integration of response representations into the next application state, $\delta : AppStates \times (Reqs \times LTypes) \rightarrow \mathcal{P}(AppStates)$. Furthermore, since the ε -NFA includes ε -transitions, for some application states the transition function may change the application state without reading an input symbol, i.e. without the system generating a resource manipulation request.

3.2 Example Web Application

In order to illustrate concepts presented in this paper, we introduce a weather forecast Web application. The resources comprising the Web application are shown in Fig. 2. The base URI of the application is <http://weather.example.com> and we identify its resources using relative addressing. The main Web page, located at `/main`, contains an `<a>` link to the details Web page and a `<script>` element pointing to a JavaScript script located at `/script`. The script periodically highlights the `<a>` link to the details page by changing the color from blue to red. The details Web page, located at `/details`, contains an `` tag pointing either to `/cloudy` or to `/sunny` depending on the current weather. Furthermore, the details page contains a `<script>` element with inline JavaScript code which uses XMLHttpRequest for periodically fetching the current temperature from `/temp` and displaying it in the Web page. Finally, the details page contains an `<a>` link pointing to the main page. The media types of the resources are `text/html` for `/main` and `/details`, `image/png` for the `/sunny` and `/cloudy`, and `text/javascript` and `text/plain` for `/script` and `/temp`, respectively. We assume that the user of the application is using a modern Web browser.

The ε -NFA model of the example Web application is shown in Fig. 3, with numbers denoting states and letters denoting input symbols. The initial state

**Fig. 2.** Example Web application**Fig. 3.** ϵ -NFA model of example Web application

0 contains a representation with an `<a>` link to the `/main` page. After a GET request is issued (**a**), the server returns a response containing the representation of the `/main` page which becomes the current state. State 1 is not steady since the representation contains a `<script>` link to `/script` which must be fetched. A GET request is issued to fetch the script (**b**) and the application then enters the steady state 2. Because the script periodically changes the color of the link pointing to the `/details` page, the application's steady state may change between states 2 and 3 without issuing a request (ϵ).

When the user follows the link to the `/details` page (**c**), the application makes a nondeterministic transition to transient states 4 and 5 because the representation contains an `` link pointing either to `/sunny` or `/cloudy`. After fetching the linked image using a GET request (**d** or **e**), the application enters a steady state 6 or 7. Furthermore, the representation of the details page contains an inline script which periodically makes a GET request for the current temperature (**f**). Since the returned temperature may have two possible values (25C or 30C), the application nondeterministically enters states 8 and 9 if the weather was cloudy, or states 10 or 11 if the weather was sunny. The script

then inserts the temperature into the page without issuing requests (ε), moving the application into states 12 and 13, or 14 and 15, depending on the weather. Because the details page contains a link to the main page, the user may at any time follow the link (a) and bring the application back to state 1.

3.3 Client-Server Style and Stateless Style Constraints

Figure 4 shows the system-level view of a RESTful system as a set of modules, their mapping to the elements of the ε -NFA and distribution between client and server components. Because client-server interaction in RESTful systems must be stateless, the *Application State* module which stores the current application state is located on the client. This is also true for modules that generate input symbols: the *Media Type Processor*, *Application-level Logic* and *Hypermedia-level Logic*. However, the transition function is divided between the client and server in order to satisfy the client-centric description of RESTful system operation in which the server is responsible only for mapping from requests to responses. Therefore, the *Request Preprocessor*, *State Integrator* and *Code-on-demand Engine* reside on the client, and only the *Request Processor* on the server.

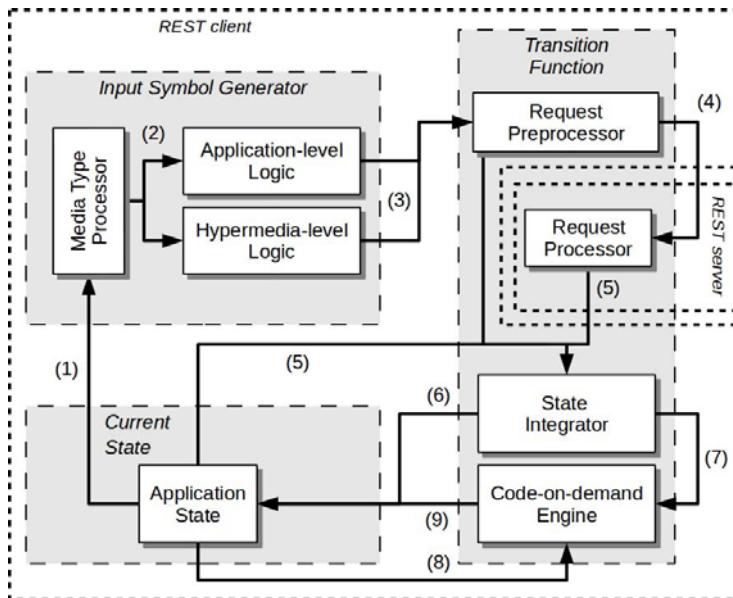


Fig. 4. Mapping of client-server and stateless REST constraints to the ε -NFA model

The interaction of the system's modules is defined as follows. The resource representations comprising the current application state are read by the *Media Type Processor* (1) in order to determine the set of available hypermedia links. For example, the `/main` page of the example Web application has the

`text/html` media type which enables that the `<a>` link to the `/details` page and `<script>` link to the `/script` script be recognized. The set of links and application state are passed to the Application-level Logic and Hypermedia-level Logic (2) so that one of the links may be chosen as the basis for the next input symbol. Hypermedia-level Logic is responsible for generating input symbols which guide the system to a steady state. Because steady states are determined exclusively from the media types of the representations in the current application state, Hypermedia-level Logic functions independently of Application-level Logic. On the other hand, Application-level Logic is responsible for generating input symbols based on application-specific goals, which are derived either from user input or from application-specific rules encoded in the module. For example, after the `/main` page has been fetched, two links may be followed, `<script>` for embedding the `/script` script and `<a>` for navigating to the `/details` page. The former link would be selected by Hypermedia-level Logic for downloading the script, while the latter link would be selected by Application-level Logic, but only in response to the user clicking on the link.

The input symbol generated by either of these modules consists of a resource manipulation request and the link type of the chosen link (3). The Request Preprocessor stores and removes the link type of the input symbol and adds a request identifier to the request before forwarding it to the Request Processor on the server (4). The server's response therefore contains the request identifier and a representation of the identified resource. Although request identifiers are a conceptual requirement for coupling responses with requests, they are not currently used on the Web since requests and responses are related through the TCP connection by which they are sent and received. The State Integrator uses the link type connected with the request, the corresponding server response and the current application state (5) to synthesize the next state (6). For example, if the `/main` page was fetched and the `/script` script was fetched afterwards via the `<script>` link, the script representation would be added to the application state. On the other hand, if the `/details` page was fetched afterwards via the `<a>` link, the received representation would replace the existing application state.

Finally, if the resource representation is an executable script, the integrator passes the script to the Code-on-demand engine for execution (7). The script may then examine and change the current application state (8) (9) without issuing requests to the server. For example, after the `/script` script is fetched and executed using a JavaScript engine, it periodically modifies the application state by changing the color of a link in the representation of the `/main` page.

3.4 Uniform Interface Style Constraint

REST is defined by four *uniform interface* constraints: identification of resources, manipulation of resources through representations, self-descriptive messages, and hypermedia as the engine of application state. In this section we formalize these constraints in the context of our model. *Resource identification* is supported in the model through resource identifiers which are used explicitly in input symbols

and application states, where an input symbol consists of a resource manipulation request and link type, where the request contains a resource identifier, a method and a representation. For example, an input symbol $IS_{\text{toDetails}}$ for navigating to `/details` in the Web application example could be represented as:

$$IS_{\text{toDetails}} = (\text{Request} : (\text{Method} : \text{"GET"}, \text{ResourceId} : \text{/details"}, \text{Representation} : \text{"}), \text{LinkType} = \text{"<a>"}) ,$$

and the application state AS_{main} of a completely loaded `/main` page as:

$$AS_{\text{main}} = [(\text{metadata} : \text{...}, \text{data} : \text{/maincontents"}), (\text{metadata} : \text{...}, \text{data} : \text{/scriptcontents"})] .$$

Due to lack of space, we do not include full listings of representation data and metadata. On the Web, metadata in general consists of HTTP headers, while the data is the body of HTTP message.

Manipulation of resources through representations is supported in the model through explicit usage of representations in input symbols and application state. One of REST's foundations is the temporally-varying mapping of resources to representations, which is supported through the nondeterminism of the transition function. For example, because the `/details` page contains an `` link to either `/cloudy` or `/sunny`, the navigation from `/main` to `/details` in the example Web application could be represented with the following transition:

$$\delta(AS_{\text{main}}, IS_{\text{toDetails}}) = \{ AS_{\text{detailsCloudy}}, AS_{\text{detailsSunny}} \} , \text{ where}$$

$$AS_{\text{detailsCloudy}} = [(\text{metadata} : [\text{mediaType} : \text{"text/html"}], \text{data} : \text{/details content with link to /cloudy"})] ,$$

$$AS_{\text{detailsSunny}} = [(\text{metadata} : [\text{mediaType} : \text{"text/html"}], \text{data} : \text{/details content with link to /sunny"})] .$$

The *self-descriptive messages* style constraint is supported in the model through stateless interaction, the limitation of using finite sets for system methods, media types, link types and link relations, and explicitly using these elements in the input symbols and application state. For example, the $IS_{\text{toDetails}}$ input symbol shown above has a link type of `<a>` while the representations in states $AS_{\text{detailsCloudy}}$ and $AS_{\text{detailsSunny}}$ are of the `text/html` media type.

Hypermedia as the engine of application state is supported in the model through the transition function which advances the system from one state to another. Specifically, the output of the transition function should be defined only for pairs of states and input symbols for which the input symbol may be derived from the current state. In other words, the current state must contain a hypermedia link used to generate the next input symbol's link type and resource manipulation request. For example, the transition function in the model of the

example Web application is undefined for state $AS_{detailsCloudy}$ and $IS_{toDetails}$ because the `/details` page does not contain an `<a>` link to itself:

$$\delta(AS_{detailsCloudy}, IS_{toDetails}) = \{\} .$$

Furthermore, we define that the initial application state is a single representation containing links to the resources which are the stable entry points for the system. For example, since the example Web application's entry point is the `/main` resource, the initial state of the model AS_{init} could be represented as:

$$AS_{init} = [(metadata : [mediaType : "text/html"], data : "HTML page with an <a> link to /main")].$$

Finally, *steady* and *transient application states* are supported in the model through accepting and unaccepting states. The acceptance of a state is determined from the media type of the first representation in the application state. For example, in the example Web application the first representation in an application state is always of the `text/html` media type meaning that all embedded resources, such as resources linked to using `` and `<script>`, should be fetched in order for the system to be in a steady state. Therefore, the state AS_{main} is accepting (steady), while the state $AS_{detailsCloudy}$ is unaccepting (transient).

3.5 Code-on-Demand Style Constraint

The *code-on-demand* style constraint is defined [2] as client-side execution of downloaded scripts together with the possibility that these scripts extend the functionality of the client. We formalize the code-on-demand constraint in the model through ε -transitions i.e. if a script executing on the client may change the application state from A to B without issuing a request, then this change is modeled with an ε -transition as $\delta(A, \varepsilon) = \{B\}$. In the example Web application, the `/script` script changes the color of the `<a>` link of the `/main` page which can be modeled as $\delta([main_{link_blue}], \varepsilon) = [main_{link_red}]$ and $\delta([main_{link_red}], \varepsilon) = [main_{link_blue}]$, where $main_{link_red}$ and $main_{link_blue}$ are the representations of the `/main` page in which the link is colored red and blue, respectively.

4 Conclusion and Future Work

The study of architectural styles is an essential part for understanding and improving information systems. As the World Wide Web is the most important global information system, the study of its foundational architectural style, Representational State Transfer (REST), is of equal importance from both a theoretical and a practical perspective. Our analysis of previous research in this field has shown that formal models of RESTful systems are unresearched, consider only few core principles of RESTful systems while ignoring others and are focused on modeling hypermedia systems in general and not RESTful systems.

In this paper we propose a formalism for modeling RESTful systems based on nondeterministic finite-state machines with epsilon transitions (ε -NFA). We

show that ε -NFAs are a natural fit for modeling REST's principles which are primarily concerned with exchange of representations using states and transitions. Specifically, the states of the ε -NFA represent the application states which the system may be in at some point of execution, where each application state is a set of resource representations. The input symbols of the ε -NFA represent the set of resource manipulation requests while the transition function models the hypermedia links between resources i.e. the request which may be issued at some state. In order to support the time-varying nature of resource representations, transitions may be nondeterministic, while ε -transitions are used to model client-side execution of code-on-demand scripts. The client-server style of REST is therefore naturally modeled by storing the current ε -NFA state and generating input symbols on the client while the transition function is divided between the client and server. The client is responsible for transforming input symbols into requests and integrating resource representations into the application state, while the server is responsible for processing requests into responses. Representation media types determine which states of the ε -NFA are accepting or not, representing respectively steady and transient states of the system. Therefore, with respect to the presented model, a system may be called RESTful if it can be represented with an ε -NFA and if sequences of generated input symbols do not lead the ε -NFA into an empty error state.

Our research gives the following insights and areas for future work. First, although we have shown an example of using the presented formalism in modeling a simple Web application, the formalism should be applied to more systems, including complex Web applications, widget-based Web applications, Web APIs and mashups. A special focus of this effort would include the modeling of composite RESTful applications. However, the goal would not be to extend the model so that it supports modeling of all properties of all systems, but rather to use it for reasoning about which properties of such systems are RESTful.

Second, it would be useful to explore the possibility of extending the presented formalism so that it explicitly accounts for currently unaddressed principles of RESTful systems. For example, this could include the layered and cacheable style constraints, and resources that map to a set of representations with different media types, with the goal of modeling content negotiation.

Third, one specific issue which may be raised is that of the finite set limitations of the presented model. Because RESTful systems are not restricted to a finite number of states or input symbols, finite-state machines are not a completely suitable model. The ε -NFA model may be relaxed so that the set of states may be infinite or even not countable or, alternately, other kinds of models for describing infinite-state machines may be used. One possible candidate are labeled state transition systems [26], a formalism similar to finite-state machines which permits that the number of states and transitions be infinite.

Fourth, we will explore practical applications of the model. One possible direction is to use the ε -NFA model as the basis for designing a framework for development of RESTful systems, and specifically RESTful Web applications.

Existing Web application development frameworks do not support the implementation of many RESTful constraints, such as the uniform interface constraint, shifting that burden to the developer. We believe this to be a direct consequence of nonexisting models with system-level mappings and that the model in this paper is simple, yet powerful, and understandable by developers. This premise is supported by the recently developed Restfulie framework [27] which imposes state transitions as the underlying application development model.

Finally, in order to avoid the state explosion problem for models of complex systems, we will consider aggregating similar states of models into a single state. However, this requires that a suitable method for aggregation be chosen. Currently, we are researching an approach based on abstracting representations into two parts: the constant application-level data and the variable set of hypermedia links. This would enable that states which correspond to the same set of representations with the same set of links and which differ only by the data would be aggregated into a single state, significantly reducing the number of states.

Acknowledgments. The authors acknowledge the support of the Ministry of Science, Education, and Sports of the Republic of Croatia through the *Computing Environments for Ubiquitous Distributed Systems* (036-0362980-1921) research project. Furthermore, the authors thank Sinisa Srblic, Dejan Skvorc, Miroslav Popovic, Klemo Vladimir, Marin Silic, Jakov Krolo and Zvonimir Pavlic from the School of Electrical Engineering and Computing, University of Zagreb.

References

1. Jacobs, I., Walsh, N.: Architecture of the World Wide Web, Volume One. W3C Recommendation, WWW Consortium (2004), <http://www.w3.org/TR/webarch/>
2. Fielding, R.T., Taylor, R.N.: Principled design of the modern Web architecture. ACM Transactions on Internet Technology 2(2), 115–150 (2002)
3. Dusseault, L., Snell, J.: PATCH Method for HTTP. In: Proposed Standard, Internet Engineering Task Force, IETF (2010), <http://tools.ietf.org/html/rfc5789>
4. Trifa, V., Trifa, V., Guinard, D., Bolliger, P., Wieland, S.: Design of a Web-based Distributed Location-Aware Infrastructure for Mobile Devices. In: 1st IEEE International Workshop on the Web of Things, Mannheim, Germany, pp. 714–719 (2010)
5. Alarcon, R., Wilde, E.: Linking Data from RESTful Services. In: 3rd International Workshop on Linked Data on the Web, Raleigh, North Carolina, USA (2010)
6. Fitzpatrick, B., Slatkin, B., Atkins, M.: PubSubHubbub protocol, <http://pubsubhubbub.googlecode.com/svn/trunk/pubsubhubbub-core-0.3.html>
7. Rosenberg, F., Currera, F., Duftler, M.J., Khalaf, R.: Composing RESTful Services and Collaborative Workflows: A Lightweight Approach. IEEE Internet Computing 12(5), 24–31 (2008)
8. Pautasso, C.: RESTful Web service composition with BPEL for REST. Data & Knowledge Engineering 68(9), 851–866 (2009)
9. Koch, N.: Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process. Ph.D. dissertation, Ludwig-Maximilians-University of Munich, Germany (2000)

10. Fernandez, F., Navon, J.: Towards a Practical Model to Facilitate Reasoning about REST Extensions and Reuse. In: 1st International Workshop on RESTful Design, Raleigh, North Carolina, pp. 31–38 (2010)
11. Rees, J.: ACTION-434: Some notes on organizing discussion on WebApps architecture. In: W3C TAG Mailing List (2010),
<http://lists.w3.org/Archives/Public/www-tag/2010Oct/0061.html>
12. ISSUE-60: Web Application State Management. W3C TAG Issues List,
<http://www.w3.org/2001/tag/group/track/issues/60>
13. Fielding, R.T.: ACTION-434: Some notes on organizing discussion on WebApps architecture. In: W3C TAG Mailing List (2010),
<http://lists.w3.org/Archives/Public/www-tag/2010Oct/0100.html>
14. Kemp, J.: AWWW and the Web interaction model. In: W3C TAG Mailing List (2010), <http://lists.w3.org/Archives/Public/www-tag/2010Jun/0034>
15. Vinoski, S.: RESTful Web Services Development Checklist. IEEE Internet Computing 12(6), 95–96 (2008)
16. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Publishing, Reading (1979)
17. Berners-Lee, T., Mendelsohn, N.: The Rule of Least Power W3C TAG Finding (2006), <http://www.w3.org/2001/tag/doc/leastPower.html>
18. Mehta, N.R.: Composing style-based software architectures from architectural primitives. Ph.D. dissertation, University of Southern California, California, USA (2004)
19. Hernandez, A.G., Moreno Garcia, M.N.: A Formal Definition of RESTful Semantic Web Services. In: 1st International Workshop on RESTful Design, Raleigh, North Carolina, pp. 39–45 (2010)
20. Decker, G., Luders, A., Overdick, H., Schlichting, K., Weske, M.: RESTful Petri Net Execution. In: Bruni, R., Wolf, K. (eds.) WS-FM 2008. LNCS, vol. 5387, pp. 73–87. Springer, Heidelberg (2009)
21. Alarcon, R., Wilde, E., Bellido, J.: Hypermedia-driven RESTful Service Composition. In: 6th Workshop on Engineering Service-Oriented Applications, San Francisco, California (2010)
22. Charlton, S.: Building a RESTful Hypermedia Agent, Part 1 (2010),
<http://www.stucharlton.com/blog/archives/2010/03/building-a-restful-hypermedia>
23. Alalfi, M.H., Cordy, J.R., Dean, T.R.: Modeling methods for web application verification and testing: state of the art. In: Software Testing, Verification & Reliability Archive, vol. 19(4), pp. 265–296. John Wiley and Sons Ltd., Chichester (2009)
24. Stotts, P.D., Furuta, R., Cabarrus, C.R.: Hyperdocuments as Automata: Verification of Trace-Based Properties by Model Checking. ACM Transactions on Information Systems 16(1), 1–30 (1998)
25. Dargham, J., Al-Nasrawi, S.: FSM Behavioral Modeling Approach for Hypermedia Web Applications: FBM-HWA Approach. In: Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services, Guadeloupe, French Caribbean, pp. 199–199 (2006)
26. Trybulec, M.: Labelled State Transition Systems. Formalized Mathematics 17(2), 163–171 (2009)
27. Parastatidis, S., Parastatidis, S., Webber, J., Silveira, G., Robinson, I.S.: The role of hypermedia in distributed system development. In: 1st International Workshop on RESTful Design, Raleigh, North Carolina, pp. 16–22 (2010)

Knowledge Spaces

Marcos Baez, Fabio Casati, and Maurizio Marchese

Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, Italy
`{baez,casati,marchese}@disi.unitn.it`

Abstract. This paper presents a set of models and an extensible social web platform (namely, Knowledge spaces) that supports novel and agile social scientific dissemination processes. Knowledge spaces is based on a model for scientific resources that allows the representation of scientific knowledge and meta-knowledge, of effective “viral” algorithms for helping scientists find the knowledge they need, and of interaction metaphors that facilitate its usage. The concept and a preliminary implementation of Knowledge spaces, in their various forms and designs, are being exploited in several different pilots.

Keywords: knowledge dissemination, social web, scientific publications.

Knowledge spaces (kspaces for short) are a metaphor, a set of models and processes, and a social web platform that help you capture, share and find scientific knowledge, in all of its forms.

The principle behind kspaces is to allow knowledge dissemination in the scientific community to occur in a way similar to the way we share knowledge with our colleagues in informal settings. The rationale behind this is that when we interact informally with a small team of colleagues dissemination is very effective. We are free to choose the best format for communicating our *thoughts* and results, we share both established results as well as latest ideas, we interact and *carry on a conversation* (synchronously or via email), we *comment* on other people's contributions and papers and observe *relations* among various contributions. Even when we remain in the domain of papers, we often find that we come to know interesting papers not by doing a web search or scan the proceedings, but because we "stumble upon" them, that is, we have colleagues pointing them to us via email or mentioning them in a conversation (along with their comments), and *knowledge spreads virally*.

Kspaces aim at providing the models, processes, metrics and tools to support this informal and social way of disseminating knowledge among the scientific community at large and via the Web, complementing the well-established method of papers published in conferences and journals after peer review. The goal is to use a web-based system to enable the capturing of these evolutionary bits of knowledge and data, however they may be expressed, as well as the capturing of ideas and opinions about knowledge, and leverage this information and meta-information to spread knowledge virally. Capturing opinions on knowledge is particularly important. The fact for example that somebody (and especially somebody we “trust”) shares a paper tells us a lot on the value of this paper, much more than a citation can do. As readers, we relate them, in our mind, with prior knowledge. When listening to a talk we think that other work is relevant to the one being presented and often we jot it down in our

own personal notes. In a world where information comes out from the web like from a hose, this knowledge about knowledge becomes essential to dissemination. Tagging, annotating and connecting the dots (linking resources in a way much more useful to science than citations) become almost as important as the dots themselves.

Kspaces support this not only by using web technologies as the basis for its implementation but by using web 1.0 and 2.0 concepts in the way scientific resources and their relationships are modeled and in the way knowledge sharing is supported. In essence, kspaces is characterized by a conceptual model and a repository for scientific resources (or for pointers to them if stored elsewhere). Resources are linked in arbitrary ways and relationships are typed and can be annotated. This is analogous to the Web, although it is oriented to linking scientific resources and to supporting (and then leveraging) relationship types and annotations. Indeed building this evolving web of annotated resources and leveraging it to find knowledge is a key goal of kspaces. The intuition is that having such web of connected knowledge can be as instrumental or even *more* instrumental (because it contains more metadata) to finding knowledge than the Web is to finding web pages. Today this web of resources is simply not there and this is part of what makes finding interesting scientific knowledge hard.

On top of this space of resources, kspaces define specific processes, permissions, and interaction modes people use to share knowledge. Kspaces manifest themselves in various forms, called *designs*, tailored at capturing different forms of scientific knowledge shared in different ways, from maintaining a library of related work, talks, datasets, etc, in an area – including our own, evolving work - to forming knowledge communities, writing and publishing (liquid) books, supporting the collection of the knowledge that emerges in the mind of attendees during a talk, and many others. It is through spaces with specific design that knowledge and meta-knowledge is collected and disseminated. The dissemination and search of knowledge over kspaces is then based on the “social interest”, on the goals of a search (e.g., related work vs introductory material), and on the meta-knowledge (e.g., tags and annotations). Kspaces, although being richer and more flexible than many existing systems, is not the first and only platform that exploits some form of social meta-knowledge to support search. Mendeley, citeUlike, and Connotea, just to name a few, all have some elements of this. We believe that the key to a successful platform here lies in how such meta-knowledge can be collected and how it is used, and here lies a key contribution of kspaces.

Accessing kspaces. Kspaces have been developed in the context of the LiquidPub project (liquidpub.org). They are the results of several attempts and failures at arriving at a model for capturing knowledge, which we initially tackled by trying to impose a specific knowledge collection mechanism (that is, a single, specific KS app). More details, demos, and applications are available at open.kspaces.net

Acknowledgements. We acknowledge the great contributions to the ideas and the code from all the liquidpub team, with particular thanks to Alex Birukou and to all our kspaces developers including Delsi Ayala, Simone Dalcastagne, Nicola Dorigatti, Lyubov Kolosovska, Muhammad Imran, Michele Lunelli, Aliaksey Minyukovich, Danil Mirilenko, Alejandro Mussi, Cristian Parra, Laura Pop. This work has been supported by the LiquidPub project (FP7 FET-Open 213360.).

Exploratory Multi-domain Search on Web Data Sources with Liquid Queries

Davide Francesco Barbier, Alessandro Bozzon, Marco Brambilla, Stefano Ceri,
Chiara Pasini, Luca Tettamanti, Salvatore Vadacca,
Riccardo Volonterio, and Srđan Zagorac

Politecnico di Milano
Department of Electronics and Information (DEI)
Piazza L. Da Vinci 32,
I-20133 Milan, Italy
`{name.surname}@elet.polimi.it`

Abstract. We demonstrate Liquid Queries, a novel user interaction paradigm for exploratory multi-domain search upon structured information collected from heterogeneous data sources. Liquid Queries support an exploratory search approach by providing a set of interaction primitives for multi-domain query formulation, result visualization and query refinement, with commands for perusing the result set, changing the visualization of data based on their type (e.g., geographical) and interacting with the remote search services.

1 Introduction

Liquid Queries are developed in the context of Search Computing (SeCo)[4], a framework for search applications that bridge the gap between general-purpose and vertical search engines. SeCo queries extract ranked information about several interconnected domains, such as “real estate”, “job” or “school”, by interacting with Web data sources which are wrapped as search services. Search Computing systems support their users in asking multi-domain queries; for instance, “where can I find a new job nearby a nice furnished flat having a good school for my children at walking distance”.

In this paper we demonstrate Liquid Queries [2], a novel interaction paradigm able to support continuous evolution, manipulation, and extension of multi-domain queries and results, so as to grant exploratory information seeking [5], according to the “search as a process” paradigm [1]. Among existing search systems, Search Computing integrates and extends concepts proposed by Kosmix¹, which performs topic-based clustering and queries on different data sources with a federated search model, Google Squared², which presents results in the shape of tables, Google Fusion Tables³, which allows users to submit structured data and provides spreadsheet-like views of individual or joined tables.

¹ <http://www.kosmix.com/>

² <http://www.google.com/squared/>

³ <http://tables.googlelabs.com/>

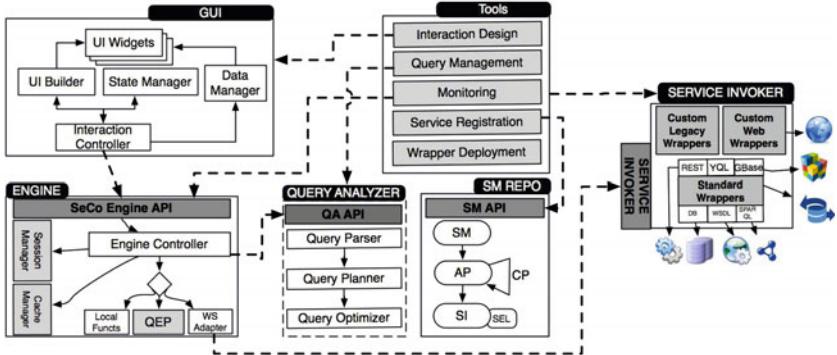


Fig. 1. The Search Computing architecture

2 The Liquid Query Interaction Paradigm

The Liquid Query life-cycle consists of four steps, namely the 1) application configuration phase, in which the expert user defines a liquid query template for a specific application; 2) a query submission phase, where the end user submit the initial liquid query; 3) the query execution phase, where a liquid result set is produced and delivered to the user interface; and 4) the result browsing phase, where results can be manipulated through appropriate interaction primitives.

2.1 Architecture

The Liquid Query demo sits on top of an architecture that covers all the phases necessary for formulating and processing multi-domain search queries (Figure 1). SeCo queries are addressed to Web data sources, including search engine APIs, community curated data sources (e.g., YQL Open Data Tables, DBpedia), domain-specific databases (e.g., Amazon, Eventful, Zillow), etc.

Data sources are registered in the system using the *Service Mart Repository*, which contains a conceptual and operational description of the search services [3]. Sources are registered as *service marts* characterized by the service name and a collection of attributes; this description is refined into one or more *access patterns*, i.e., logical signatures that specify whether each attribute is either an input or an output in the service call; output attributes are tagged as ranked if the service produces results ordered on the value of that attribute. Access patterns are then refined into service interfaces, which include the name and the endpoint of a concrete search service.

Queries enter the system at the Liquid Query graphical user interface. A *Query Analyzer* translates the query in a Query Execution Plan (QEP), a graph of low-level components that specifies the activities to be executed (e.g., the service calls), their order of precedence, and the strategy to execute joins. The actual service invocation is managed by the *Execution Engine*, which supervises the interaction with the service interfaces to access Web APIs and databases. The results of service calls are accumulated by the engine, which builds progressively the combinations constituting the query response. These combinations are

submitted back to the Liquid Query interface for visualization and user interaction. Through the GUI, users can then manipulate the result set. Each user manipulation (e.g., filtering, re-rank, exploration of a new domain) produces a new query, which is analyzed and executed.

2.2 Liquid Query Interaction

Liquid Query provides a set of interaction primitives and controls over the query engine for dynamically changing the results presented to the user. Such controls include: *service exploration*, to inspect the Service Mart repository and find the most suitable services; *search expansions* which enables a controlled form of exploration where the user can select one or more combinations of interest and ask for novel information on some of the included objects (e.g., chosen a Concert, ask for information about the recent News associated with it); request of *more results* from all services or from a selected subset (these commands need interaction with the search back-end system); *sorting* of results, *clustering*, *grouping*, hiding or showing of result properties, reordering of attributes and services in the result table, and *query history* management. Liquid Queries support query expansion and result tracking, giving the user the possibility to move “forward” and “backward” along the exploration history. The Liquid Query paradigm supports interaction through several result set visualization paradigms.

3 Demonstration Scenario and Highlights

The demonstration provides a detailed walk through of all the steps needed to prepare, execute, and refine a multi-domain query over Web data sources with Liquid Queries. The demo shows the registration of service marts, access patterns and physical service interfaces for several types of data sources. Based on the registered services, the demonstration continues by showing how users can explore the available information space by inspecting the Service Mart repository, select a data source and query it, iteratively select the best answers and then inspect additional properties which are reachable from such answers. An example demonstration scenario assumes that an end user wants to move in the Silicon Valley, where she wants to find a new job as a Java developer, also considering the availability of fully-furnished, close-by flats and good schools. Users can additionally look for doctors close to the candidate location, for pieces of news associated to a given employer and for good restaurants nearby.

Finally, we show how users can perform information exploration by switching among several result visualization paradigms. The simplest one is the *Tabular View*, where combinations are presented as rows, sorted with respect to the global ranking functions. Service mart attributes are presented as columns. The so-called *Atom View* (Figure 2(a)) is devised to highlight the local population and ranking individual service marts, which are less visible in the Tabular View, by showing the object’s name (or any suitable identifier), while more properties can be asked for separately. Within an Atom view, users can select combinations (in which case all objects forming the currently selected combination are highlighted)



Fig. 2. (a) Atom view, which highlights both individual objects and combinations, with their local and global rankings; (b) Visualization of objects and combinations based on geographical location, with explicit ranking information

or objects (in which case all the combinations it belongs to and associated objects are highlighted). Types of result data can be used to enable type-dependent visualizations of objects and their relationships: for instance, the geographic coordinates of the involved objects can be exploited to represent them in a map (Figure 2(b)): each object is represented by a given marker, and the local ranking (e.g., the price of the House or the rating of the School) is represented by the size of such marker. A combination is then represented by a set of different markers, which are highlighted when the combination is selected. The demo will also feature the well known *XYPlot* visualization paradigm, where users can inspect results according to two attributes values, respectively rendered on the X and Y axis of the graph; each object is represented with a marker in the Cartesian space, and users can dynamically modify the attributes assigned to each axis.

References

1. Baeza-Yates, R., Raghavan, P.: Chapter 2: Next generation web search. In: Ceri, S., Brambilla, M. (eds.) Search Computing. LNCS, vol. 5950, pp. 11–23. Springer, Heidelberg (2010)
2. Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P.: Liquid Query: Multi-domain Exploratory Search on the Web. In: WWW 2010: 19th International Conference on World Wide Web, pp. 161–170. ACM, New York (2010)
3. Campi, A., Ceri, S., Maesani, A., Ronchi, S.: Designing service marts for engineering search computing applications. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 50–65. Springer, Heidelberg (2010)
4. Ceri, S., Brambilla, M. (eds.): Search Computing - Challenges and Directions. LNCS, vol. 5950, pp. 3–10. Springer, Heidelberg (2010)
5. Kuhlthau, C.C.: Inside the search process: Information seeking from the user's perspective. Journal of the American Society for Information Science 42(5)(5), 361–371 (1991)

Model-Based Dynamic and Adaptive Visualization for Multi-domain Search Results

Alessandro Bozzon, Marco Brambilla, Luca Cioria, Piero Fraternali,
and Maristella Matera

Politecnico di Milano, Dipartimento di Elettronica e Informazione
P.zza L. Da Vinci, 32. I-20133 Milano - Italy
`{name.surname}@polimi.it`

Abstract. Search systems are becoming increasingly sophisticated in their capacity of building result sets that are not mere lists of documents but articulated combinations of concepts retrieved from different domains. This paper investigates the models for the result sets and the visualization spaces, and model-to-model transformations to dynamically suggest optimized visualizations for multi-domain search results.

1 Introduction

Past years have seen an evolution in the way search engines, and more generally information seeking applications, deliver responses to user's information needs. There has been a shift from supporting simple keyword-based search tasks over flat collections of documents, where the effectiveness of a retrieval system is concentrated in the capacity of showing most relevant results first, to addressing more complex information needs and constructing more articulated responses.

This is already visible in mainstream search engines, which are now capable of recognizing quite a large amount of concepts in the input keywords (people, cities, events) and provide a customized representation of results, e.g., by including maps, photographs and videos, and concept-dependent data, like tourism information for a retrieved city. More sophisticated approaches to result visualization are also provided by vertical search applications, which exploit domain knowledge to optimize the display of retrieved results (see for example <http://www.wanderfly.com>).

Our work addresses the problem of automating the construction of result visualizations for multi-domain search tasks, where results are ranked combinations of objects with typed attributes and relationships. The core idea is to model both the *data set* and the *visualization space*, and to construct a *model-to-model* mapping that dynamically determines which visualization to use, based on static and dynamic result properties (e.g., data types and attribute value distribution).

2 Visualizations for Multi-domain Search

In our approach, the problem of *multi-domain search* is defined as the computation and presentation of results to queries over multiple Web data sources that

return ranked lists of objects. A typical multi-domain query can be: “*Find combinations of hospitals and doctors specialized in the treatment of a given disease, ranked based on the hospital rating and on the scientific impact of the doctor*”.

Answering multi-domain queries requires a processing architecture as the one being implemented in the Search Computing (SECO) project [2]. In the SECO architecture, a client tier, the *Liquid Query Graphical User Interface* (LQ GUI) [1], allows the user to formulate queries instantiating pre-registered search application skeletons, declaring the available data sources with the signature of the access methods, and connection paths that “join” a source to another one. The currently implemented LQ GUI has a fixed set of data visualizations (table, atom view, and parallel coordinates) and offers commands for perusing the result set (hiding and showing attributes, expanding the query by joining in more data sources, asking for more results from one or all data sources, and changing the rank criterion). The server tier of the architecture then comprises a *Service Repository*, where external data sources can be wrapped and registered using a variety of technologies (Rest, WSDL, YQL, SPARQL, and GBASE), and a *Query Processor*, which decomposes the user query into service calls, and then sends an execution plan to an *Runtime Engine* in charge of invoking services and assembling results efficiently.

The work described in this paper aims at equipping the LQ GUI module with the capability of automatically suggesting visualizations to the user, based on the features of the current result set and on the available visualization templates; both aspects are encoded as models. The result set data is thus analyzed on the fly and the visualization is adapted to the characteristics of the retrieved objects (e.g., since hospitals are geo-referenced, results are displayed on a map).

3 Overview of the Visualization Process

The adaptive visualization process, with its inputs and outputs, is shown in Figure 1. The goal is to determine the best mapping from data providers of the *result set model* (*attribute values, object instances and combinations of objects*) to data renderers of the *visualization model* (*axes and visual clues* that make up *templates*) so that the result set is visualized in a way that is adapted to the distribution of objects and combinations in the result set, the types of the object attributes, and the preferences about which information to show first.

The output view is decided in consecutive steps. *Dynamic Analysis* collects statistics on result set data that may impact visualization (e.g., range and density of attribute values, homogeneity of attribute value distribution, selectivity of join conditions among combination objects). In parallel, *Static Analysis* extracts from the result set model visualization priorities of attributes according to the characteristics of the type, the suitability of an attribute to identify an object, and relative importance of the attribute information content (e.g., the address of a hospital could be relatively more important than its name). As a second step, *Data Mapping* is performed: based on the specification of visualization templates provided by the abstract presentation model, heuristic rules inspired by classic works on data

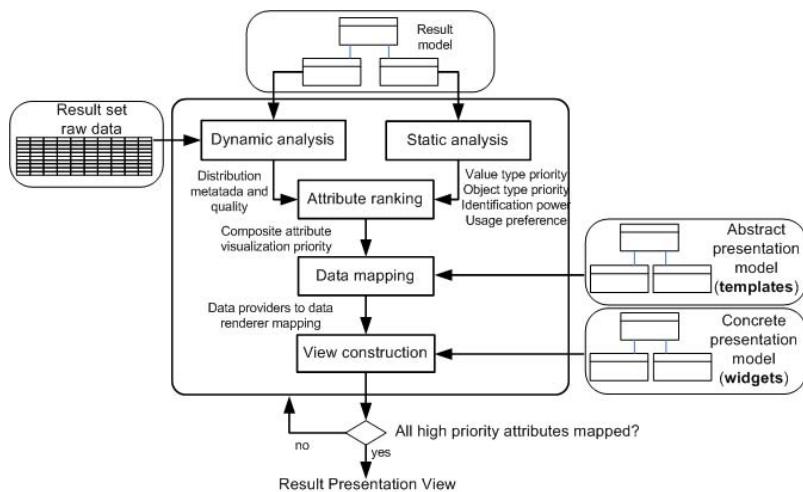


Fig. 1. Overview of the process of adaptive and dynamic result visualization

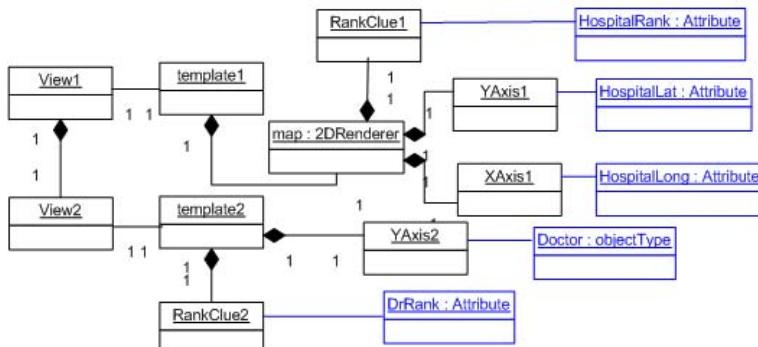


Fig. 2. A rendered view (top) and its abstract model (bottom)

visualizations (e.g., [3]) are employed to calculate a matching between the sorted list of attributes and the elements in the available templates: each template receives a suitability score and the top-ranking template is chosen for visualization. Finally, *View Construction* converts the chosen abstract template into a concrete view, by replacing abstract data renderers with concrete widgets from the concrete visualization model (e.g., a map template is concretely implemented as a Google map view with overlaid HTML 5 elements). The process can be recursive: if attributes with priority above a threshold could not be assigned to a template, a sub-view can be created by invoking the same process. Typically, this is done on an object-by-object basis, to create sub-views that can be displayed on demand (e.g., pop-up windows with the details of a doctor not displayed on the map). When all important attributes are mapped, the (possibly nested) view is instantiated and added to the LQ GUI, to be directly rendered or suggested to the user.

Figure 2 shows an example of rendered view (top) and of the corresponding abstract model (bottom). The example deals with hospitals, ranked by score, and doctors working at them, also sorted by score. The view consists of a map template and of a nested subview. The map template has two axis, for geographical coordinates, and a visual clue dimension, for a numerical attribute. The subview consists of a list template, with one vertical axis and a visual clue for a numerical attribute (the hospital rank). The bottom part of Figure 2 shows the visualization model resulting from the mapping process, which highlights the mapping of data providers to element of the visual template. The latitude and longitude of hospital objects are associated with the axis of the map template, and the hospital rank to the visual clue. The axis of the list template in the subview is mapped to the object instances of the doctor object type, and the visual clue to the doctor's specialty and rank.

4 Conclusions

Based on the visualization process illustrated in this paper, we have implemented a first prototype that adds dynamic and adaptive result visualizations to the SECO LQ GUI component, by configuring and instantiating at run-time a number of concrete presentation widgets implemented as HTML and Javascript view components. Our future work will concentrate on improving the current implementation, on the provision of more visualization templates and concrete widgets, and on the fine tuning of heuristic mapping rules, also with the help of usability studies and user assessment experiments.

References

1. Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P.: Liquid query: multi-domain exploratory search on the web. In: Proc. of WWW 2010, pp. 161–170 (2010)
2. Ceri, S., Abid, A., et al.: Search computing: Managing complex search queries. IEEE Internet Computing 14(6), 14–22 (2010)
3. Chi, E.H.-h.: A taxonomy of visualization techniques using the data state reference model. In: INFOVIS, pp. 69–76 (2000)

A Constraint Programming Approach to Automatic Layout Definition for Search Results

Alessandro Bozzon, Marco Brambilla, Laura Cigardi, and Sara Comai

Politecnico di Milano

Department of Electronics and Information (DEI)

Piazza L. Da Vinci 32,

I-20133 Milan, Italy

{alessandro.bozzon,marco.brambilla,sara.comai}@polimi.it

Abstract. In this paper we describe a general framework based on constraint programming techniques to address the automatic layout definition problem for Web search result pages, considering heterogeneous result items types (e.g., web links, images, videos, maps, etc.). Starting from the entity type(s) specified in the search query and the result types deemed more relevant for the given entity type, we define an optimization problem and a set of constraints that grant the optimal positioning of results in the page, modeled as a grid with assigned weights depending on the visibility.

1 Introduction

Search engines represent one of the most important classes of applications that support the user information seeking process [5]. As the amount and the different kinds of information grow, also user requirements change, and search engines need to adapt accordingly. For instance, recently Web searchers started looking directly for objects of interest instead of Web pages that describe such objects. Search engines adapted to this new demand by improving the management of *named entities* specified within the queries and by providing well organized result pages that comprise different result types (e.g., maps, news, images, etc.) based on the type of entity involved in the query. The different result types may be produced by different search engines (image search engines, blog search engines, and so on), but then the results need to be aggregated in an optimal unified layout. For instance, a query involving an actor will produce a very different set of result types with respect to a query involving a city: the former case will feature videos, pictures, and blogs, while the latter will include maps, touristic reviews, and local news (see Figure 1 for an example of result page returned for the query “Washington”). Several problems arise in this scenario: how to map named entities to the most significant result types, how to select the best mix of items to show, and how to define the best layout of the result page. In this paper we propose a general framework that exploits constraint programming techniques to automatically compose a *search engine result page* (SERP): result items are positioned in a fixed page grid, based on a score function to be maximized and on

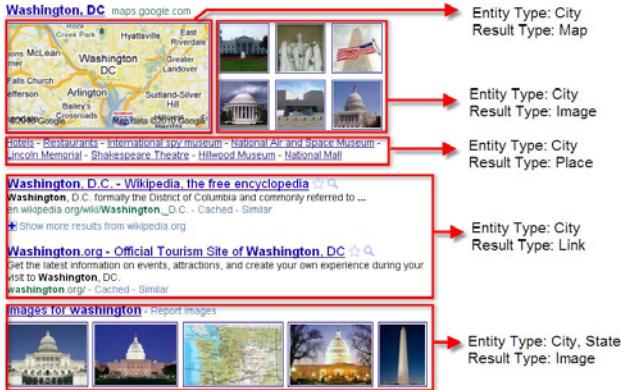


Fig. 1. Excerpt of the result presentation page for the query “Washington”

a set of structural constraints imposed by the typical search engine behaviour. We provide an overview of the behaviour of our system (Sect. 2) and the description of the automatic page layout composition approach (Sect. 3).

2 Search Engine Model

The behaviour of our system basically consists of the following phases [2]: 1) the **Query submission** phase, in which the user submits the query typically through a search form, by specifying some keywords, also including named entities; 2) the **Entity recognition** phase, in which the system first identifies the types of entity provided by the user in the query; 3) the **Result type selection** phase, in which the system defines which result types should be considered in the results; 4) the **Search Services Invocation**, in which the appropriate search services are invoked for each of the identified result types; and finally 5) the **SERP composition**, in which the results are positioned in the page. In this paper we focus on the latter phase.

3 SERP Model

We defined a conceptual model of the objects relevant for the problem, including: 1) the **page model**, defined as a grid of cells with an assigned level of importance as shown in Figure 2.a (in particular, we assigned the importance according to the "Golden Triangle" [4] pattern, that defines the most valuable area of the page as the top-left one – the importance levels are represented by shades of gray); 2) the **result type**, defined as the media type that is retrieved by a search engine and referenced in the page (e.g., image, web link, map, ...); 3) a (typed) **result block** (Figure 2.b) defined as a set of homogeneous results of a given result type, boxed within a shape of given size in terms of number of cells that it covers (e.g., a map occupies an area of 3x2 cells, while each web

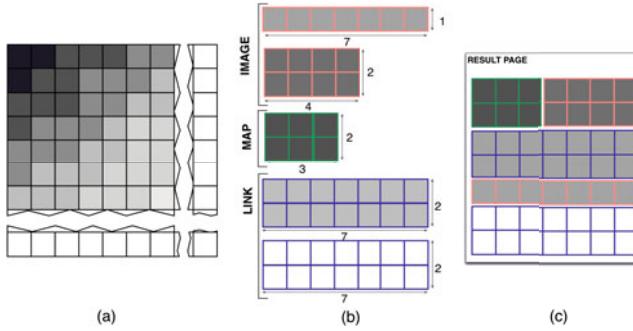


Fig. 2. Example of (a) page grid with importance levels, (b) some weighted result blocks, and (c) a possible block allocation

link occupies an area of 7x1 and blocks aggregating two items are considered). Result types are assigned with a score that depends on the entity type (e.g., for cities, maps have a higher score than pictures, while for actors maps have a very low score and pictures are extremely relevant). Result blocks are assigned a score that depends both on the result type they represent and on the result instances that are actually retrieved by the search engine for the specific query. This is exemplified in Figure 2.b, where blocks have been colored according to the corresponding result types and instances (for example, in Figure 2.b we have two image blocks, with different scores).

4 SERP Layout Composition

Given the SERP model described above, the SERP layout composition consists in an optimization problem with constraints induced by some structural rules in the positioning of blocks, specified to restrict the space that a block can occupy (e.g., the image block can be place on the top, in the middle, or at the end of the page; sponsored links must occupy a fixed position, related searches too, and so on). Examples of constraints include:

```

Constraint C1 = and(eq(SLx, 8), eq(SLy, 0)); //Sponsor link block pos. fixed at (x=8, y=0)
Constraint C2 = and(geq(L1x, 0), leq(L1y, 4)); //L1 block starts in one of the first 5 rows
Constraint C3 = distanceEQ(Ay, By, 1); //Blocks A, B must be positioned one after the other
  
```

The constraint satisfaction problem (CSP) describing the structural constraints for page positioning is modeled and then solved through a standard CSP solver [1]. The best layout is selected by maximizing the sum of the products between the block scores and the weight of the occupied cells:

$$\max \sum_i (score_{ResultBlock(i)} * \sum_j weight_{OccupiedCells(j)})$$

This function is actually modeled by means of some heuristic constraints that position blocks with high scores in cells having high weights, so as to avoid generating layouts that are for sure suboptimal. This reduces the computation cost of determining the optimal layout among the possible ones, without increasing significantly the computation cost of the constraint solver. Once the optimal

layout is produced by the constraint solver (i.e., all the coordinates of the result blocks are fixed), a concrete html page can be produced, where positioning is defined by means of appropriate CSS rules (see the example in Figure 2.c).

5 Evaluation

A preliminary analysis has been performed to evaluate two aspects: comparison with existing solutions and added value perceived by the users. The *comparison* consists in comparing the page layouts of our approach with the ones of the major search engines (Google, Yahoo! and Bing). We assessed that our approach can deal with all the layouts produced by the various search engines without any conceptual change; the only change is in the set of constraint rules: on average, we needed to add or change 3 rules to align to Google, 6 rules for Yahoo!, and 5 rules for Bing. The *perceived value analysis* consisted in a user test aiming at comparing different constraint rule configurations. The evaluation involved 16 users, who were asked to provide their preferences among 4 versions of the result pages, for all the entity types. The analysis concluded that there is an high perceived value associated with the optimization algorithm, while different version of the constraint rules were perceived basically the same: the 3 optimized versions got around 30% of the score each, while the other got 10%.

6 Conclusions and Future Work

We discussed a framework for the automatic layout of search results, which is a more and more critical issue in industrial search engines despite being overlooked in the research community. We proposed a constraint based solution to the optimization problem of positioning the search items in the page at the purpose of maximizing the result page quality perceived by the user. As future work we plan to tackle queries involving several entity types at the same time [3] and to add additional variables to the problem, such as information coming from runtime statistics on contents, personalization and contextualization of the search task.

References

1. Choco library, <http://choco.emn.fr/>
2. Bozzon, A., Brambilla, M., Comai, S.: A Characterization of the Layout Definition Problem for Web Search Results. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6428, pp. 150–159. Springer, Heidelberg (2010)
3. Ceri, S., Brambilla, M. (eds.): Search Computing - Challenges and Directions. LNCS, vol. 5950, pp. 3–10. Springer, Heidelberg (2010)
4. Hearst, M.A.: Search User Interfaces, 1st edn. Cambridge University Press, Cambridge (2009)
5. Kuhlthau, C.C.: Inside the search process: Information seeking from the user's perspective. Journal of the American Society for Information Science 42(5)(5), 361–371 (1991)

Adaptive Mobile Web Applications: A Quantitative Evaluation Approach

Heiko Desruelle, Dieter Blomme, and Frank Gielen

Ghent University – IBBT,

Dept. of Information Technology – IBCN, Ghent, Belgium

{Heiko.Desruelle,Dieter.Bломme,Frank.Gielen}@intec.ugent.be

Abstract. The rapidly growing market of mobile devices has set a need for applications being available at anytime, anywhere, and on any device. Although this evolution provides users an unprecedented freedom, developers are facing the challenges caused by mobile device fragmentation. Current application development solutions are insufficiently optimized for the high diversity of mobile platforms and hardware characteristics. In this paper, we propose a novel approach for the development of mobile applications. An adaptive application composition algorithm is introduced, capable of autonomously bypassing fragmentation related issues. This goal is achieved by introducing a quantitative evaluation strategy derived from the Logic Scoring of Preference (LSP) method.

1 Introduction

Mobile is a powerful mass medium, with a greater reach and faster growth than any other known media type [1]. The mobile evolution has resulted in a general need for applications and services being available at all times. However, only few technologies are currently capable of supporting the development and delivery of a single application to many types of devices. This barrier is a result of the heavily fragmented mobile landscape. In order to cover a sustainable share of the mobile market, applications need to be made adaptable to various combinations of hardware, operating systems, APIs, etc.

In response to this challenge, the use of the web as an application platform is gaining momentum. Device independent web technologies such as HTML, CSS and JavaScript offer application developers an unprecedented market reach. Nevertheless, even with the use of standardized web technology, efficiently managing mobile fragmentation remains an important research topic. As browser implementations still contain many variability points, true mobile convergence is not to be expected any time soon [5].

Within this context, the goal of our research is to create automated adaptability processes for the development and delivery of mobile web applications. We present an application composition algorithm as a means of supporting mobile applications to autonomously resolve fragmentation related issues. The proposed algorithm aims at offering a robust and future proof approach for the flexible composition of web applications based on the individual capabilities of the mobile device.

2 Capability-Driven Progressive Enhancement

Since the early days of web engineering, developers have tried to cope with the differences between browsers. Graceful degradation is a widespread design strategy that focuses on providing optimal support for the most advanced browsers. Less capable browsers are only considered during the last development phase. This approach often results in a poor stripped-down version. The graceful degradation methodology expects users to just upgrade their browser when the degraded version does not fit their needs. However, for most mobile devices upgrading the default browser is not an option.

Progressive enhancement (PE) reverses the graceful degradation approach and aims at maximizing accessibility over browsers with different capabilities [8]. PE tries to achieve this goal by forcing developers to take the less capable devices into account from the very start of the development process. First, a basic markup document is created, providing an optimal experience for devices with the lowest common denominator (LCD) of available capabilities. Incrementally, one or more layers of structural, presentational, and behavioral enhancements are added in function of the browser's specific capabilities.

The PE methodology can be used in a mobile context to tackle fragmentation related issues. However, when turning the theoretical approach into actual practice, a number of important challenges come into play. Today, the use of externally linked resources (e.g. CSS, or JavaScript files) is the most common practice for selecting appropriate enhancement layers. This limits the number of detectable variability points, as browsers will only check for coarse-grained styling and scripting support. Compared to desktop browsers, the mobile ecosystem contains far more combinations of browsers with graded CSS and JavaScript support. To provide optimized usability, PE should also reckon with the different interaction methods and hardware characteristics offered by mobile devices. As we will discuss in the following section, the creation of a viable mobile PE solution requires an application development approach that supports the use of more fine-grained enhancement layers.

3 Adaptive Application Composition Algorithm

As a means to address the wide variety of mobile characteristics, we introduce a quantitative evaluation algorithm derived from the Logic Scoring of Preference (LSP) method [3]. The algorithm is designed to support fine-grained progressive enhancement and is capable of suggesting a stack of layers that optimally fits the user's mobile device. LSP is a quantitative decision method, assisting decision makers in the evaluation, comparison, and selection of complex hardware and software systems. The method has shown its use in various domains, especially concerning situations with large and complex solution spaces.

To evaluate a set of candidate solutions, LSP starts by assessing n individual performance variables. These variables define the n properties that an ideal solution is expected to have. As the algorithm deals with complex decision problems,

most candidate solutions will not perfectly match the preset criteria. Nevertheless, such candidates should not be rejected, as their overall evaluation might still lead to an acceptable solution. LSP addresses this issue by taking into account how well a candidate matches the different performance variables. For each variable i , a degree of suitability $E_i \in [0, 1]$ is calculated. In order to attain these scores, LSP requires a predefined mapping function for each performance variable [4].

After obtaining the elementary degrees of satisfaction, all individual matching scores are to be combined into one objective overall suitability score. This aggregated score is used to determine the best-matching candidate. LSP supports the definition of an aggregation network, expressing the specific conjunction or disjunction relationships between individual scores. The standard aggregator mechanism is based on the superposition of fundamental Generalized Conjunction Disjunction (GCD) [2]. GCDs enable the specification of aggregations in terms of 17 graded combinations of conjunction and disjunction and are frequently implemented by use of Weighted Power Means (WPM). This approach allows an evaluator to precisely couple the mutual importance of individual suitability degrees. The calculated aggregation network results in an objective overall suitability score E , which is a combination of one or more WPMs using the individual suitability degrees as input parameters. After calculating E for each of the candidates, conclusions regarding the best-matching solution can be drawn by selecting the candidate with the highest overall suitability score.

LSP has the ability to flexibly, yet objectively, evaluate systems under various circumstances. This quality can be used as a basis for the adaptive composition of mobile applications. We propose a modification of the LSP method that supports the adaptive composition of mobile web applications. In this case, all possible sets of progressive enhancement layers are considered candidate solutions. Each candidate is individually evaluated by matching it to the mobile device's capabilities (e.g. available interaction methods, web technology support, etc.). The stack of enhancement layers with the highest score is then selected and applied to the mobile web application.

Table 1. Boolean mobile mapping function. Only perfect matches are scored.

Interaction capability	Match
Touch	0 %
Stylus	100 %
Joystick	0 %
Click wheel	0 %

Table 2. Fuzzy mobile mapping function. Also grading less-than-perfect matches.

Interaction capability	Match
Touch	75 %
Stylus	100 %
Joystick	30 %
Click wheel	10 %

Incorporating the LSP method in a mobile context requires the definition of mobile-relevant mapping functions. The functions specify the similarity between performance variables and the actual device capabilities. To illustrate the concept, both Table 1 and 2 contain the implementation of a mapping function that compares the performance variable "stylus interaction" with a device's interaction method. The function in Table 1 uses Boolean logic, which implies

that only a perfect match is scored. The one in Table 2, on the other hand, uses fuzzy logic [6]. The latter approach makes much better use of the available scoring interval by also grading the less-than-perfect matches. This example highlights the importance of developing carefully thought through mapping functions. Efforts in the usability area from groups such as the W3C Mobile Web Best Practices Working Group can be used to generate sets of mobile mapping functions [7].

4 Conclusion and Future Work

In this paper, we introduced an algorithm in support of developing and delivering adaptive mobile web applications. The proposed method can serve as a basis for developers to create and maintain a single version of their mobile application, without being limited by fragmentation related issues. Our adaptive application composition algorithm is based on a quantitative evaluation algorithm derived from the Logic Scoring of Preference method. The proposed approach enables the automated and fine-grained progressive enhancement of web applications. The process is entirely driven by the characteristics of the user's device, in order to provide an optimal user experience.

While the extensive evaluation of our approach has yet to be carried out, initial testing of prototype implementations showed promising results. Future work includes the validation of our proposed approach as well as the extension of our algorithm towards supporting real time application request handling.

Acknowledgments. The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7-ICT-2009-5) under grant agreement number 257103.

References

1. Ahonen, T.: *Mobile As 7th of the mass media. Cellphone, Cameraphone, iPhone, Smartphone*. Futuretext, London (2008)
2. Batyrshin, I., Kaynak, O., Rudas, I.: Generalized conjunction and disjunction operations for fuzzy control. In: Proc. of 6th European Congress on Intelligent Techniques and Soft Computing EUFIT 1998, pp. 52–57. Verlag Mainz, Aachen (1998)
3. Dujmovic, J.J.: A method for evaluation and selection of complex hardware and software systems. In: Proc. of 22nd Int. Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems, pp. 368–378 (1996)
4. Dujmovic, J.J., De Tre, G., Van de Weghe, N.: LSP suitability maps. *J. Soft. Computing* 14, 421–434 (2010)
5. Frederick, G.R., Lal, R.: *The future of the mobile web*. In: *Beginning Smartphone Web Development*, pp. 303–313. Springer, Heidelberg (2009)
6. Ross, T.J.: *Fuzzy logic with engineering applications*. Wiley, Chichester (2004)
7. World Wide Web Consortium: Mobile Web Best Practices Working Group, <http://www.w3.org/2005/MWI/BPWG>
8. Wells, J., Draganova, C.: Progressive enhancement in the real World. In: Proc. of 18th Conference on Hypertext and Hypermedia HT 2007, pp. 55–56. ACM, New York (2007)

A Personality Mining System for Automated Applicant Ranking in Online Recruitment Systems

Evanthia Faliagka¹, Lefteris Kozanidis¹, Sofia Stamou^{1,3}, Athanasios Tsakalidis¹, and Giannis Tzimas²

¹ Computer Engineering and Informatics Department, University of Patras, Patras, Greece

² Department of Applied Informatics in Management & Economy, Faculty of Management and Economics, Technological Educational Institute of Messolonghi, Messolonghi, Greece

³ Department of Archives and Library Science, Ionian University, Greece

{faliagka,kozanid,stamou}@ceid.upatras.gr,
tsak@cti.gr, tzimas@cti.gr

Abstract. In the last decades the explosion of ICT has opened up new avenues regarding peoples' accessibility to new job opportunities. Current technological advances in conjunction with people's online presence provide a great opportunity to automate the recruitment process and make it more effective. In this paper, we propose a novel approach for improving the efficiency of e-recruitment systems. Our approach relies on the linguistic analysis of data available for job applicants, in order to infer the applicants' personality traits and rank them accordingly. To showcase the functionality of our method, we employed it in a web based e-recruitment system that we implemented.

Keywords: e-recruitment, sentiment analysis, recommendation systems, AHP.

1 Introduction

E-recruitment systems have seen an explosive expansion in the past few years [1] allowing HR agencies to target a very wide audience. The price paid is the uncontrolled increase of unqualified applicants. This situation might be overwhelming to HR agencies that need to allocate human resources for manually assessing the candidate resumes and evaluating the applicants' suitability for the positions at hand. To alleviate this problem, several e-recruitment methods have been proposed, the majority of which rely on standard IR and web mining approaches for matching candidates to open positions [2]. Existing methods, although useful, suffer from the discrepancies associated with inconsistent CV formats, structure and contextual information.

In this paper, we propose a novel approach in job applicants ranking based on the Analytic Hierarchy Process, AHP [3] and the automatic extraction of applicant personality measures. The latter is based on the linguistic analysis of textual data pertaining to applicants' profiles available on Web 2.0 sites. Our approach is implemented in a web based employer-oriented e-recruitment system and tested in real-world data, in a pilot scenario designed in collaboration with Novartis Hellas HR department.

2 Method

Applicants' selection in the proposed e-recruitment system is based on a predefined set of criteria that are assessed on a numerical scale. We focus the present study on the exploitation of 4 complementary criteria, namely: *Education* (in years of formal academic training), *Work Experience* (in years), *Loyalty* (average number of years spent per job) and *Personality*. Each criterion has a distinct contribution in the selection process, as dictated by Novartis Hellas HR. At the ranking process we use the AHP, which allows diverse elements to be compared to one another in a rational and consistent way. Objective selection criteria (i.e. all but candidate personality) are directly extracted from the applicants' LinkedIn profiles. On the other hand, for assessing the candidate's personality, we exploit textual data available for the candidate. To implement our method, we essentially considered job applicants with a LinkedIn account and an active blog. Our method was field-tested in a pilot scenario, which involved the recruitment of a set of applicants. Our system was employed to extract candidate rankings based on the aforementioned set of criteria, and identify the top candidates to pass to the next phase of the recruitment process.

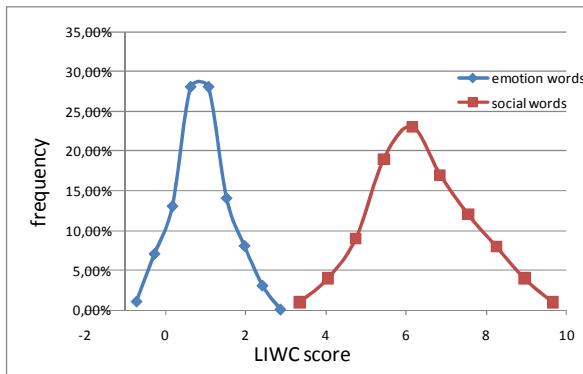


Fig. 1. Social and emotion words' frequency distribution

2.1 Personality Mining

Applicant personality traits are considered critical in most job positions, but are overlooked in existing e-recruitment systems. In our study extroversion is a crucial personality characteristic for candidate selection. Previous work indicates that blogs contain textual features reflecting the author's personality [4]. Specifically, it has been established that extrovert people use many social words and positive emotion words, and few negative emotion words. Thus, to quantify the candidate extroversion we use LIWC text analysis program, to measure the fraction of words in a candidate's blog that fall in these categories (denoted as social, posemo, negemo). It has been shown in [5] that there are significant correlations between word categories used in LIWC and the corresponding personality traits. In order to train our system, we used a corpus of 100 Greek blogs, and extracted the LIWC scores per word category for each blog.

Then, we estimated the social word frequency distribution, as well as the emotion word frequency distribution. The emotion category is obtained as the difference between posemo and negemo category, essentially penalizing the use of negative emotion words. The distributions are shown in Fig. 1, with LIWC scores clustered in intervals with a length of 0.5 and represented by their mid value. These distributions serve as a basis to calculate the *standard score* of applicants' personality dimensions (social and emotion), based on the distance of their LIWC scores from the corresponding mean value.

2.2 Applicant Ranking

The overall applicant score is obtained from individual scores in the selection criteria, using the Analytical Hierarchy Process (AHP). Each criterion has a different weight in the candidate selection, according to the requirements of the job position. Thus, the first step in the AHP process is to make pairwise comparisons of the selection criteria, forming the matrix $A = (a_{ij})_{n \times n}$. Parameter a_{ij} expresses the relative importance of criteria i and j and it is provided by an expert recruiter. Then the normalized eigenvector of the matrix is computed, which serves as the weight vector.

The next step is performing pairwise comparisons of candidates with respect to each criterion. Five 15×15 matrices are computed, one per criterion, with the ratio of the criteria scores per candidate pair. Finally, the normalized eigenvector of the matrices are calculated, obtaining five local priority vectors. The overall score of each candidate (also termed global priority, or rank) is computed as a linear combination between weight and local priority vectors.

3 Experimental Results

The proposed e-recruitment method was implemented as a web application and tested in a real recruitment scenario. It uses LinkedIn API to extract the applicant's objective criteria and LIWC system with a Greek dictionary to assess the applicant's extroversion from his blog posts. Each applicant simply logs to the system with his LinkedIn account credentials and enters his blog URL. The system then estimates the applicants' overall rank using AHP and outputs the top candidates. The system was tested in a pilot scenario with 15 candidates applying for a sales position.

Table 1. Local and Global priorities

Weight vector	Social (0.33)	Emotion (0.33)	Education (0.11)	Work (0.17)	Loyalty (0.07)	Global priorities
Candidate 1	0.15	0.08	0.07	0.04	0.09	0.097
Candidate 2	0.03	0.04	0.11	0.07	0.08	0.052
...						
Candidate 15	0.06	0.08	0.09	0.08	0.06	0.074

The weight vector used is seen in the first row of Table 1. It is obvious that the first two (personality related) criteria are the most important, as indicated by the recruiter. The rest of Table 1 shows the local and global priorities for 3 of 15 candidates due to

space limitations, calculated with AHP. Finally, in Fig. 2 we represent the candidates with circles positioned in a 2D plane based on their personality scores, while the circle radius is proportional to the candidate overall score. Average values of X and Y axis are clearly marked in Fig. 2. It is evident that most highly ranked candidates are clustered in the top right quadrant (i.e. with high emotion and social scores), which attests that our tool assigned high ranks to candidates with the desired personality.

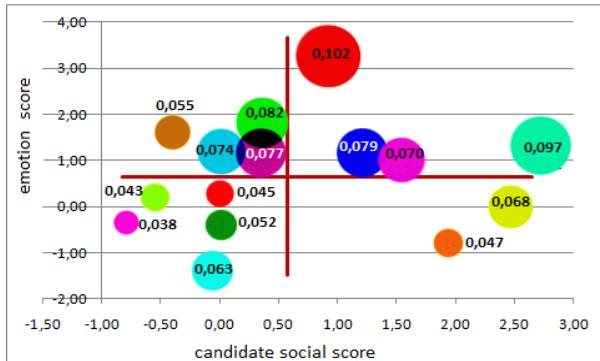


Fig. 2. The candidates' personality and overall scores

4 Conclusions

In this paper we have presented a novel approach for recruiting and ranking job applicants in online recruitment systems. The application of our approach reveals that it is effective in identifying personality profiles for job applicants and thus rank them accordingly. We are currently improving our method with additional personality features as well as further sources of candidate textual data and in the future we plan to deploy it in a large-scale e-recruitment application.

References

1. De Meo, P., Quattrone, G., Terracina, G., Ursino, D.: An XML-Based Multiagent System for Supporting Online Recruitment Services. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 37, 464–480 (2007)
2. Kessler, R., Béchet, N., Torres-Moreno, J., Roche, M., El-Bèze, M.: Job Offer Management: How Improve the Ranking of Candidates. In: Rauch, J., Rás, Z.W., Berka, P., Elo-maa, T. (eds.) *ISMIS 2009*. LNCS, vol. 5722, pp. 431–441. Springer, Heidelberg (2009)
3. Saaty, T.L.: How to make a decision: The analytic hierarchy process. *European Journal of Operational Research* 48, 9–26 (1990)
4. Gill, J.A., Nowson, S., Oberlander, J.: What are they blogging about? Personality, topic, and motivation in blogs. In: Proc. of AAAI ICWSM (2009)
5. Pennebaker, J.W., King, L.: Linguistic Styles: Language Use as an Individual Difference. *Journal of Personality and Social Psychology* 77, 1296–1312 (1999)

Development of the Evaluation Form for Expert Inspections of Web Portals

Andrina Granić¹, Ivica Mitrović², and Nikola Marangunić¹

¹ Faculty of Science, University of Split, Nikole Tesle 12, 21000 Split, Croatia
`{andrina.granic,nikola.marangunic}@pmfst.hr`

² Arts Academy, University of Split, Glagoljaška bb, 21000 Split, Croatia
`ivica.mitrovic@umas.hr`

Abstract. Web portals are a special breed of web sites, providing a large and diverse user population with a blend of information, services and facilities. Due to the lack of heuristics for the design and evaluation of general portals, we have conducted an experimental work to create research instrument for expert inspections. A set of 36 equally-levelled guidelines was developed to measure the hypothetical usability categories which reflect the four most distinctive elements of the contemporary web design process. The *Portal Guidelines* for expert inspections prove valuable to both novice inspectors and HCI experts, but further research will be required to validate the findings of this study.

Keywords: usability, expert inspection, Portal Guidelines, web portal.

1 Introduction

The idea of web portal is to collect information from different sources and create a single point of access to information, functions and services that are relevant to person's work or personal interests [1]. Due to specifics of portals as web sites, here primarily addressing their complex and hybrid structure and media specificities along with diversity of user population, their tasks and workflows, usability issues are crucial for their design. Yet many current web portals suffer from problems related to low usability since in many portal projects usability is an afterthought or is completely ignored in others. Usability assessment is often seen as too expensive, difficulty to accomplish, too specialized or something to address after testing all the "functionality" [2]. Consequently the efficient evaluation of web portals, and websites in general [3], has become a point of concern for both practitioners and researchers.

Horizontal information (broad-reach and news) portals, also called general, are nowadays the most visited Croatian web sites [4]. Whether such portals do indeed reach their aim of facilitating users' access to diverse resources at the same time targeting the entire Internet community and, if so, to which extent, needs further investigation. To evaluate how easy to use horizontal information portals are, we have conducted a series of experiments that employed a range of usability assessment methods [5]. Unlike most studies related to web portal evaluations which usually rely on user-based surveys (*cf.* [3] as well), our approach employed both empirical and

analytical assessment. Such methodology is in line with general assumption that we should not rely on isolated evaluations, but instead make use of complementing usability techniques and of people with different expertise whom should be involved [6]. Our experience indicated that the chosen research instruments, measures and methods for web portal usability testing were adequate. Conversely, although showing considerable potential, analytical assessment raised some concerns. With the intention to employ expert inspections to complement rather than replace usability testing, the analytical evaluation was additionally checked up.

Due to the lack of guidelines/heuristics for the design and evaluation of general web portals, we have conducted an experimental work to develop a heuristic evaluation form for horizontal information portals. The work presented in this paper is based on our previous study when Nielsen's usability heuristics, as a set of ten key principles [7], was explained and adjusted to the general portal usage. As additional clarifications to each principle, a series of auxiliary guidelines concerning portal design were also provided, cf. e.g. [8]. Comprehensive quantitative and qualitative analysis of the acquired data revealed that some of the Nielsen's principles showed poor applicability in the web portal context. As a result, a set of seven general principles was prepared, each supplement with reduced and more helpful series of guidelines. Once again, obtained data was quantitative and qualitative analysed, taking into consideration relevant information acquired not only for the principles, but for all additional auxiliary guidelines. At this point our goal was to identify the most relevant heuristics/guidelines for web portal evaluation in order to create a heuristic evaluation form for the expert inspection of general portals.

2 Heuristics for Expert Reviews

Portal Guidelines, as a helpful research instrument for expert inspections was developed to measure the hypothetical usability categories which reflect the four most distinctive elements of the contemporary web design process. The categories also reflect actual design practice in general portals' development specifically their information architecture (INFO; 15 guidelines), navigation (NAV; 11 guidelines), layout/visual identity (VIS; 9 guidelines) and interactivity i.e. web specific practices (INT; 9 guidelines). Detailed quantitative and qualitative analysis of the guidelines was performed in our previous published experimental work. We have started with more than 100 guidelines and after comprehensive empirical evaluations the number of guidelines was reduced. As a result, a list of 44 guidelines was defined. Table 1 depicts examples of guideline statements organized in four categories. To explore reliability and validity of the evaluation form, the list was given to a group of 31 "instant/novice" [9] inspectors, chosen among computer science graduate and postgraduate students, and experienced practitioners from the field of web design.

General portal *tportal* (www.tportal.hr) was evaluated against the set of guidelines. A 5-point Likert-type scale asked the inspectors to rate the degree to which they agree with the guideline statements. Inspectors were asked to fill out an on-line version of the heuristic evaluation form. After two weeks of data collection, all 31 responses were gathered and analyzed.

Table 1. Examples of guideline statements and their placement in four categories

Guideline category	Guideline statement
INFO	Thematic categories are clearly and simply organized
NAV	The navigation (horizontal or vertical) is positioned in consistent way
VIS	Commercials are avoided from the central part of the portal/screen
INT	Long and complicated registration is avoided

To estimate the internal consistency reliability of the scores, the Cronbach alpha coefficient was calculated for the initial set of 44 guidelines based on the sample of 31 novice inspectors. Classical item analysis was conducted on the items of the form to determine whether some items were negatively affecting the reliability of the scales. Elimination of eight “weakest” items resulted in reliability increase from 0.60 to 0.83. Because the Cronbach’s alpha values were above the conventional level of 0.7 [10], the set of guidelines is considered to exhibit adequate reliability.

The first step in the exploratory factor analysis was to estimate the number of factors in the heuristic evaluation form. The Kaiser-Gutman criterion ($\text{eigenvalue} > 1$) indicated that there are more than four factors in the evaluation form. A principal factor analysis with varimax rotation (with Kaiser Normalization procedure) was conducted to ensure that guidelines for the same construct measure the particular category, while the guidelines for the other construct measure another one. Four factors that have explained 52.7% of variance in all items were extracted. The rotated factor matrix showed all the guidelines loaded on the particular latent constructs. Overall, the factors’ structure in the factor analysis was not in agreement with the hypothetical structure of the developed *Portal Guidelines*.

3 Discussion and Conclusion

This paper briefly reports on the development of the *Portal Guidelines* as a valuable research instrument for the expert inspections of web portals. The designed heuristic evaluation form was created to measure the hypothetical usability categories which reflect actual work practice and characteristic components of the current web design process. At the beginning of this study of horizontal information web portals we assumed that the initial set of 44 guidelines could be grouped into four categories that reflect today's web design practice. After analyzing the internal guidelines' consistency reliability, we have obtained 36 guidelines that have shown reliability in the context of the research.

We have analyzed the potentially applicable situation with four categories of guidelines. Principal component analysis and comparison with the hypothetical categories has revealed that the guidelines are unevenly distributed in the four component categories. In fact, significant distinctiveness between categories could not be confirmed; specifically categories do not appear to be homogeneous – guidelines from these component categories could not be coupled in a reasonable/logical way. Therefore, we could assume that the factor analysis has not confirmed our hypothesis that the proposed set of guidelines could be categorized according to such criteria. On the contrary, the set of *Portal Guidelines* should be taken into account altogether. For

that reason, the developed evaluation form is composed of 36 equally-levelled guidelines that should provide an insight into the overall usability of the portal. The additional analysis of individual guidelines which provided especially good or bad results could further interpret the specific elements of general web portal usability.

Finally we could conclude that the evaluation form for the expert inspection of general web portals has been successfully developed; the created *Portal Guidelines* are usable for both novice inspectors and HCI experts. A lot of efforts have gone into the direction of specializing general usability heuristics and validating the new proposed heuristic set. However, it would be also valuable to understand whether the defined evaluation form leads the inspectors to identify severe problems and if some categories of problems are privileged while others are neglected. The severity of the identified problems in particular is an important indicator about how good the heuristics are guiding the inspectors in the analysis of the applications, thus in problem discovering. Additionally, it would be interesting to compare the results of the inspection (the set of identified problems) with the results of an evaluation performed through a different set of heuristics or even a different assessment method.

Acknowledgments. This work has been carried out within project 177-0361994-1998 *Usability and Adaptivity of Interfaces for Intelligent Authoring Shells* funded by the Ministry of Science and Technology of the Republic of Croatia.

References

1. Beringer, J., Lessmann, C., Waloszek, G.: SAP AG – Generic Portal Pages – What Do Most Portals Need(May 21,2001), http://www.sapdesignguild.org/editions/edition3/generic_pages.asp
2. Jensen, J.J., Skov, M.B., Stage, J.: The WPU Project: Web Portal Usability. ERCIM News 78 (2009)
3. Chiou, W.-C., Lin, C.-C., Perng, C.: A strategic framework for website evaluation based on a review of the literature from 1995–2006. Information & Management 47, 282–290 (2010)
4. Digitalna Mreža (2011), <http://www.digitalnamreza.com/>
5. Granić, A., Mitrović, I., Marangunić, N.: Exploring the Usability of Web Portals: A Croatian Case Study. International Journal of Information Management (2010), doi:10.1016/j.ijinfomgt.2010.11.001
6. Sears, A., Jacko, J.: The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications. In: Human Factors and Ergonomics, 2nd edn. (2008)
7. Nielsen, J.: Heuristic Evaluation. In: Nielsen, J., Mack, R. (eds.) Usability Inspection Methods, pp. 25–64. John Wiley and Sons Inc., New York (1994)
8. Sharp, H., Rogers, Y., Preece, J.: Heuristics for Websites (2007), http://www.id-book.com/catherb/Website_heurs.php
9. Bolchini, D., Garzotto, F.: Quality and Potential for Adoption of Web Usability Evaluation Methods: An Empirical Study on MILE+. Journal of Web Engineering 7(4), 299–317 (2008)
10. Nunnally, J.C.: Psychometric theory. McGraw-Hill, New York (1978)

WebSoDa: A Tailored Data Binding Framework for Web Programmers Leveraging the WebSocket Protocol and HTML5 Microdata

Matthias Heinrich¹ and Martin Gaedke²

¹ SAP AG, SAP Research Dresden, Germany
matthias.heinrich@sap.com

² Department of Computer Science, Chemnitz University of Technology, Germany
martin.gaedke@informatik.tu-chemnitz.de

Abstract. The data binding pattern is an established technique to couple user interface (UI) elements and data objects. Various markup languages (e.g. Microsoft XAML, Adobe MXML) integrate advanced data binding concepts in order to ease application development. However, the HTML standard does not embrace means for data binding although being the Web markup language supported by millions of Web programmers. Therefore, we propose a standard-compliant WebSocket-based Data Binding (WebSoDa) framework. The WebSoDa framework synchronizes data objects and UI elements by orchestrating a Microdata-based data binding language as well as a client-side and a server-side messaging component. Thus, developers may speed up the tedious task of implementing binding associations in Web applications.

1 Introduction

In the last decade, the trend to move desktop applications to the Web continued due to the lean upgrade process, the ease of consumption and the broad device support [4]. However, bringing the desktop experience to Web applications is a demanding task. Especially challenging is the application development requiring server-side updates (e.g. stock ticker or twitter feed). The integration of server-side updates encompasses various tasks such as implementing messaging hubs, specifying a messaging format and realizing an update UI mechanism. Realizing these tasks in a distributed environment translates to a repetitive and time-consuming development activity. Therefore, we propose the WebSoDa framework which empowers developers to specify bindings in a minimal markup language. Thus, the complexity exposed by several low-level programming tasks is wrapped in a concise and declarative vocabulary. Consequently, programmers may rapidly realize two-way data binding associations.

In this paper, we proceed with a brief discussion of the state of the art in section 2. Sections 3 and 4 describe the distinct framework components and outline their interaction. In section 5, we summarize the benefits of WebSoDa.

2 State of the Art Data Binding Frameworks

Currently, various data binding frameworks exist offering distinct capabilities. Their main objective is to keep the UI and the model in synch while featuring a lightweight binding language. The following overview discusses these frameworks in the light of the Web development domain, which requires standard-compliance and support for distributed systems including a lean bi-directional communication protocol.

Desktop Binding Frameworks: They are widely adopted, because they have proven to speed up the reoccurring development task of coupling UI elements and data objects. Two prominent examples are the Microsoft Windows Presentation Foundation [6] and the Java JFace toolkit [5]. However, these frameworks are not applicable to distributed systems, such as the Web.

Rich Internet Application (RIA) Binding Frameworks: After succeeding in the desktop market, the binding concept was adopted in the Web domain. RIA frameworks such as Microsoft Silverlight or Adobe Flex introduced binding features [1]. Both frameworks offer markup languages (XAML, MXML) to define bindings. Nevertheless, choosing a RIA technology significantly narrows the addressable market since applications require a dedicated browser plug-in.

JavaScript-based Binding Frameworks: Another approach to setup data bindings is facilitated through various popular JavaScript frameworks (e.g. Prototype, YUI) [7]. They address the HTTP limitation of allowing solely client-side communication requests by leveraging expensive polling techniques (AJAX). Additionally, their binding syntax is based on pure JavaScript code rather than on declarative markup. Thus, the approach implies a steep learning curve since Web programmers have to get familiar with HTML, JavaScript and a server-side programming language.

3 The WebSoDa Architecture

In order to simplify the cumbersome task of implementing data bindings, we have developed the WebSoDa framework. The standard-compliant framework consists of three essential building blocks: a language to define bindings in HTML5 as well as two messaging components executing synchronization calls.

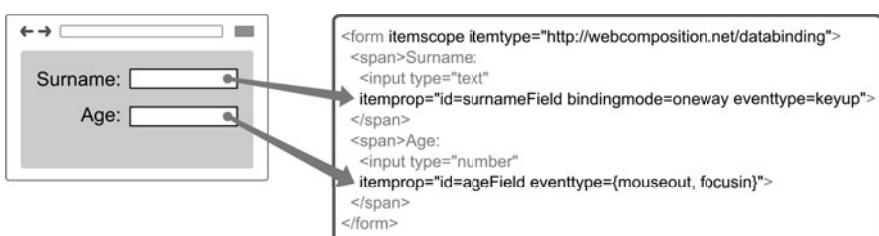


Fig. 1. A rendered HTML5 form and the associated client-side binding markup

The data binding language builds upon the Microdata specification [3] which is the default HTML5 extension mechanism. It offers three crucial attributes (`itemscope`, `itemtype`, `itemprop`) to describe annotation elements called items. While the `itemscope` attribute defines the reach of an item, the `itemtype` distinguishes the item by attaching a type URL. The item itself may contain several properties which represent key-value pairs. To create a new item property the `itemprop` keyword is used.

Figure 1 depicts the application of Microdata annotations. A form-container exposes two input elements which accept textual user input. An `itemscope` attribute creates a new typed item valid within the opening and the closing form-tag. Furthermore, two properties are defined with keys encapsulating the binding expressions. The corresponding values are the inputs provided by the end-user.

The data binding language is a minimal set of attributes. First an identifier named `id` has to be defined. Secondly, an attribute `bindingmode` expresses whether UI changes are only propagated to the model (oneway) or if model changes are also reflected on the UI (twoway). The third attribute `eventtype` specifies a set of JavaScript events which trigger the client to propagate changes to the server. All attributes are specified within the `itemprop` value field. In Fig. 1 two client-side data binding associations are defined using different binding modes and distinct update events.

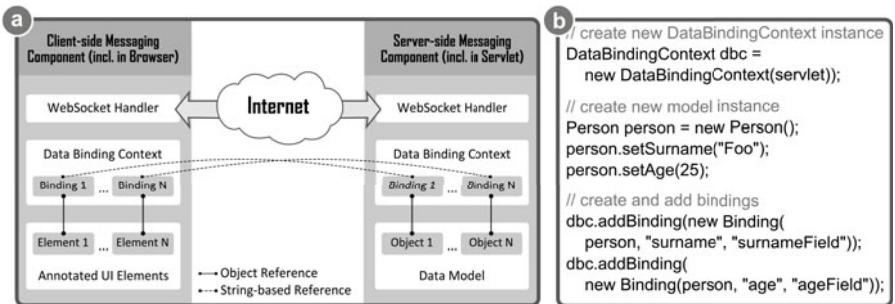


Fig. 2. The WebSoDa architecture and an example of a server-side binding expression

Besides defining a binding language, the WebSoDa framework features a client-side and a server-side messaging component depicted in Fig. 2a. These components communicate over the bi-directional WebSocket protocol [2] which is currently standardized by the IETF. The WebSocket protocol was chosen because of its native two-way communication support, its standard-compliance, the broad browser support and the minimal protocol overhead.

The client-side messaging component is packaged in an external JavaScript file named WebSoDa.js. According to Fig. 2a it is comprised of a WebSocket handler, a data binding context and the HTML5 description containing the annotated UI elements. Once the browser loads the HTML5 as well as the JavaScript definition, the data binding context registers all annotated UI elements and creates

a binding object for each. As soon as the user changes the input of a registered form field, the WebSocket handler assembles an update message which is sent to the server-side messaging component. This component retrieves the attached server-side binding which is identified by the provided *id* attribute. After selecting a binding, the UI change can eventually be propagated to the referenced model object. Updates can also be triggered by the server. In case of a model change, a message will be sent to the client and the associated field will be updated. Currently, the server-side component is implemented as a Java Servlet. The Java source code to setup a server-side data binding is illustrated in Fig. 2b.

4 Online Demo

A screencast showing the process of setting up the data binding associations, the client-side JavaScript and the server-side Java source code are available at <http://vsr.informatik.tu-chemnitz.de/demo/WebSoDa/>.

5 Conclusion

In this paper, we described a WebSocket-based data binding framework which fosters the development efficiency by integrating declarative binding expressions in HTML5. A client-side messaging component parses the binding expressions and automatically establishes bi-directional connections to the server-side model. Thus, Web developers may profit from the data binding concept without introducing a new technology. In contrast to the traditional form-processing approach, the proposed framework supports a two-way binding reflecting model changes instantly on the UI. Furthermore, bindings are highly configurable with respect to triggering events and binding modes. Besides offering flexibility, the lightweight framework embraces Web standards. Consequently, it only requires a state of the art Web browser supporting WebSockets and Microdata. Hence, a broad range of applications might benefit from the WebSoDa framework.

References

1. Deitel, P.: Internet & World Wide Web: How to Program. Prentice-Hall, Englewood Cliffs (2007)
2. Hickson, I.: The WebSocket protocol draft-76 (2010),
<http://tools.ietf.org/html/draft-hixie-thewebsOCKETprotocol-76/>
3. Hickson, I.: HTML Microdata - W3C Working Draft (2011),
<http://www.w3.org/TR/microdata/>
4. Jazayeri, M.: Some trends in web application development. In: 2007 Future of Software Engineering, FOSE 2007, pp. 199–213. IEEE Computer Society, Washington, DC, USA (2007)
5. McAffer, J., Lemieux, J.-M., Aniszczyk, C.: Eclipse Rich Client Platform. Addison-Wesley Professional, London (2010)
6. Nathan, A.: Windows Presentation Foundation Unleashed. Sams (2006)
7. Orchard, L.M., Pehlivanian, A., Koon, S., Jones, H.: Professional JavaScript Frameworks: Prototype, YUI, ExtJS, Dojo and MooTools. Wrox (2009)

Towards User-Centric Cross-Site Personalisation

Kevin Koidl, Owen Conlan, and Vincent Wade

Knowledge and Data Engineering Group (KDEG), Trinity College Dublin,
School of Computer Science, Dublin, Ireland
`{Kevin.Koidl,Owen.Conlan,Vincent.Wade}@cs.tcd.ie`

Abstract. Personalisation on the web is mostly confined to Websites of online content providers. The main drawback of this approach is the missing consideration of the users previous cross-site browsing experience resulting in an often fragmented browsing experience. This paper introduces a service driven architecture for user-centric personalisation in online cross-site tasks. We introduce the proposed architecture and results of an initial experiment evaluating cross-site personalisation across separately hosted open-source Web-based Content Management Systems (WCMS).

Keywords: Cross-Site Personalisation, Adaptive Hypermedia Systems, Web-based Recommender Systems, Open-Source Web-based Content Management Systems (WCMS).

1 Introduction

Personalisation on the web is mostly confined to specific Websites of content providers and not cross-site. It can be argued that the main reason for this shortcoming is the need for Website owners to encourage users to remain within their Website as long as possible. The expected outcome of a prolonged user visit is an increase in revenue either during the visit or in future visits [1]. The user on the other hand benefits from Website specific personalisation by receiving personalised recommendations, such as purchase related items, that may be of interest to the user. However this approach cannot assist the user in online tasks which require cross-site browsing, such as exploring product related information across enterprise and user generated Websites. The result of this fragmented browsing experience can increase user frustration through repetitive query usage within the different Websites [2]. To unify the fragmented browsing experience both the need of the user (freely browsing across the web) and the need of the content provider (encouraging the user to stay on the Website as long as possible) needs to be addressed.

2 The UNITE Architecture

The UNITE (UNIfied Task-based browsing Experience) architecture is designed to assist users in complex online tasks which are short-term and cross-site, such as in Online Customer Care scenarios. This is achieved through a third-party Adaptive

Feature Service (AFS) interfacing with the different Websites the user browses across. The main approach of UNITE is to unify terms related to Web pages the user has browsed across. This unified term model can then be used to send personalised recommendations to interfacing Websites reflecting the overall browsing experience of the user. Typical examples of personalised recommendation usage are link and content recommendation [4]. The advantage for the user is a more unified browsing experience across the web and for Website owner the possibility to tailor a more personal and customer specific Website experience. This approach reflects research related to Open Corpus Adaptive Hypermedia [3] and consists of following elements:

Term Identification Service: The main purpose of the Term Identification Service is to identify terms related to the current Webpage the user is viewing. Terms can either be retrieved directly from the interfacing Website (e.g. though an existing taxonomy or folksonomy) or through term extraction tools such as Yahoo JQL table¹. In a second stage the extracted terms can be used to receive related terms through external knowledge services such as WordNet² or openCalais³. Once an initial term-based taxonomy is created the service can train text analytics tools such as Weka⁴ to create related taxonomy terms for Websites which either have no related terms or for which the terms are not in the scope of the previously collected terms.

User Model Repository: The User Model consists of terms related to the current task of the user and which were identified by the Term Identification Service. The User Model can be enriched with addition properties such as preferred content type or language preferences.

Strategy Repository: Depending on the User Model properties the Strategy Model Repository can identify suitable strategies to create cross-site personalised recommendations.

Scrutiny Interface: To ensure user trust in the recommendations provided the user can view all models and adaptive decisions.

RESTful Service Layer: The UNITE architecture implements a RESTful service layer for client communication.

WCMS Module Extensions: Based on the mostly flexible and simple extensibility mechanisms of Web-based Content Management Systems different modules can be deployment at run-time to enable cross-site personalised recommendations.

3 Experimental Results

To provide a realistic real world scenario the initial experiment was based on Online Customer Care tasks in which users explore information related to the Symantec

¹ <http://developer.yahoo.com/search/content/V1/termExtraction.html>

² <http://wordnet.princeton.edu/>

³ <http://www.opencalais.com/>

⁴ <http://www.cs.waikato.ac.nz/ml/weka/>

Norton 360 product range. The goal of the experiment was to study the perceived usefulness of the approach. For this two system pairs were deployed; one non-adaptive WCMS system pair and one adaptive WCMS systems pair. Each WCMS system pair consisted of two Drupal⁵ based WCMS with one hosting structured manual content and the other hosting user generated forum content. Furthermore the Drupal based WCMSs were extended with modules allowing both the interfacing with the third-party service and the usage of term based personalised recommendation for link annotation. The experiment was conducted online with 36 volunteering participants from Trinity College Dublin and the University of South Australia. Each participant was asked to provide information about their knowledge in adaptive systems and the Symantec product range. After this the participants were asked to conduct two out of four real world customer care tasks. The tasks were assigned based on the Latin Square design. Each participant conducted one task in the adaptive system pair and one task in the non-adaptive system pair. After each task the participants were asked to fill out a System Usability Score (SUS) questionnaire. Finally, two concluding questionnaires were presented to the participants.

The pre-questionnaire indicated that most participants had limited knowledge in the usage of both Norton 360 products and adaptive systems. Furthermore most participants stated that they are consulting online manual and forum information on a regular base and using search engine technology frequently in searching for task related information. The SUS score resulted in 68.67 for the adaptive system pair and 62.58 for the non-adaptive systems pair. It can be argued that both scores are an indication for a general acceptance of the experimental system. However, the difference in scores is too small to make a clear conclusion on usability differences. In relation to discussing the quantitative data execution time, content views and query count was recorded. The average execution time was 24 minutes and 37 seconds (SD 05:35) in the adaptive system pair and 22 minutes and 24 seconds (SD 05:08) in the non-adaptive system pair. In relation to content views the adaptive system pair recorded an average of 116 content views (SD 32) and the non adaptive system pair an average of 96 content views (SD 20). Finally in relation to query count the adaptive system pair recorded 429 queries (260 in the manual hosting Website and 169 in the forum hosting Website) and the non-adaptive system pair 339 queries (203 in the manual hosting Website and 136 in the forum hosting Website). In order to investigate if the increased number of content views and queries resulted in a higher quality in tasks answers further analysis of the results needs to be conducted.

The qualitative data provided by the final two questionnaires is based on a 4-point⁶ LIKERT scale (1=strongly disagree, 2=disagree, 3=agree, 4=strongly agree). In relation to the usefulness of the link annotations when browsing across both Websites (manual and forum) 7 strongly agreed, 13 agreed, 11 disagreed and 1 strongly disagreed. Asked if the adaptive recommendations were relevant in relation to the content 2 strongly agreed, 19 agreed, 2 disagreed and 1 strongly disagreed. Asked if the participants were satisfied with the performance, assistance and guidance of the adaptive web system 6 strongly agreed, 18 agreed, 7 disagreed and 1 strongly

⁵ <http://drupal.org/>

⁶ To ensure the user provides a positive or negative tendency a 4-point scale instead of a 5-point scale was used.

disagreed. It can be argued that the participants mostly agreed that the systems approach was useful and relevant.

4 Conclusion and Future Work

This poster abstract introduced UNITE, a cross-site web personalisation architecture towards cross-site personalisation on the open web. Selected findings of an initial experiment were presented. Future evaluation will be addressing different aspects of the overall architecture in order to gain a more in dept understanding of cross-site personalisation and which techniques are most useful.

Acknowledgments

This research is supported by the Science Foundation Ireland (Grant 07/CE/I1142) as part of the Centre for Next Generation Localization (www.cngl.ie) at Trinity College Dublin.

References

- [1] Rose, S., Hair, N., Clark, M.: Online Customer Experience: A Review of the Business-to-Consumer Online Purchase Context. *International Journal of Management Reviews* 13, 24–39 (2011)
- [2] Feild, H.A., Allan, J., Jones, R.: Predicting searcher frustration. In: Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 34–41. ACM, Geneva (2010)
- [3] Brusilovsky, P., Henze, N.: Open Corpus Adaptive Educational Hypermedia. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) *Adaptive Web 2007*. LNCS, vol. 4321, pp. 671–696. Springer, Heidelberg (2007)
- [4] Koidl, K., Conlan, O.: Engineering Information Systems towards facilitating Scrutable and Configurable Adaptation, pp. 405–409. Springer, Hannover (2008)

Tool Support for a Hybrid Development Methodology of Service-Based Interactive Applications

Christian Liebing, Marius Feldmann, Jan Mosig, Philipp Katz,
and Alexander Schill

Technische Universität Dresden, Department of Computer Science,
Institute for Systems Architecture, Computer Networks Group
01062 Dresden, Germany

{christian.liebing,marius.feldmann,jan.mosig,philipp.katz,
alexander.schill}@tu-dresden.de

Abstract. In recent years promising approaches that provide the graphical development of service-based interactive applications were presented, but they still lack of simple platform-independent modeling and variability. We address this issue by exploiting the concept of service annotations to establish a hybrid development methodology that relies on a novel concept named temporal annotations to specify the applications navigation flow. To facilitate the development methodology, we present a graphical authoring tool.

1 Motivation

Developing service-based interactive applications manually is a time-consuming, cost-intensive and potentially error-prone task. To facilitate their development, two model driven approaches have emerged in recent years, which differ heavily in the underlying methodology and complexity of the development process.

On one hand, the Servface approach [1] follows the paradigm of service composition at the presentation layer and enables the end-user development of service-based interactive applications for a variety of platforms by composing Web services (WS) based on their dynamically generated frontends enhanced by UI-related annotations, which can be attached to an annotable WS element.

However, the approach has several shortcomings when it is applied to more complex development scenarios. Firstly, by focusing on end-user development, the approach enables modeling of rudimentary form-based applications only. Secondly, the development methodology demands the user to select a target platform at the beginning of the development process, whereby the resulting model cannot be transformed to applications running on various target platforms.

On the other hand, there is the expressive, task-driven development approach [2] of Paterno. This approach uses the notation of ConcurTaskTrees (CTT) [3] as a technology-independent task model to specify the temporal relations between user and system tasks. A platform-independent description of the abstract User

Interface (UI) forms the starting point of the approach. This representation is transformed into various platform-dependent concrete UIs and finally mapped to an implementation specific representation. However, the practical usability of this approach is reduced due to the variety of models and the need for a manual binding of abstract system tasks and concrete WS operations.

A common shortcoming of both mentioned approaches is their lacking support for variability and the associated re-development of distinctive applications.

Due to the sketched drawbacks, we present a hybrid light-weight but powerful approach that supports platform-independent, annotation-based and graphical modeling. It relies on one model, facilitates the WS binding and delivers comparable results, but does not cover all application scenarios. Subsequently, our demonstration solely shows an authoring tool, which supports the hybrid development methodology of service-based interactive applications, and does not discuss any underlying concepts in detail.

2 Temporal Annotations Plugin for Eclipse (TAPE)

Service-based interactive applications are characterized by two aspects: firstly, their functionality is entirely encapsulated behind well-defined service interfaces and secondly, graphical user interfaces enable human interactions with one or more services. Due to the application modeling on basis of WS and annotations, a suitable methodology must provide a rapid and simple applications development, customization and modification and furthermore different versions with little effort. Moreover, the expressiveness in regards of the resulting applications should fulfill state-of-the-art requirements for interactive applications.

To meet the requirements of developing service-based interactive applications and to eliminate the shortcomings of the existing approaches, we developed a hybrid approach that does not use annotations solely for describing UI-related information but also for specifying the navigation and data flow of an application. To achieve the consistent use of annotations, we extended the existing Servface annotation model [4] through the use of application-specific annotations. Initially, we analyzed the temporal relations of Paternos CTTs, whether they are potential candidates to specify the applications navigation flow and then added the specified temporal annotations to the Servface model. Due to our extensions, there is no need for a self-contained model, which stores all information.

To facilitate the development process, we developed a graphical authoring tool¹ by using the Eclipse platform and the frameworks GEF² and EMF³. We decided to develop a plugin, which has its own perspective due to the well-known look and feel that minimizes the learning curve for application developers. However, the plugin can be easily transformed into a RCP- or RAP-based application.

The internal model that is invisible from users perspective provides the distinction between UI-related and application-specific annotations to simplify the

¹ <http://www1.inf.tu-dresden.de/~s6334199/tape/>

² Graphical Editing Framework - <http://www.eclipse.org/gef/>

³ Eclipse Modeling Framework - <http://www.eclipse.org/emf/>

independent specification. As a result, the authoring tool produces a description that may contain UI-related and applications specific rather temporal annotations and provides their transformation together with the appropriate WS descriptions into executable interactive applications for various target platforms.

The UI of the authoring tool is structured into three important areas (depicted in Fig. 1): 1. The *Service overview* shows the available services including the associated operations, 2. The *Editor view* represents the working view and consists of an annotation tool bar to specify graphically temporal relations between WS operations, and 3. The *Project explorer* provides a project overview.

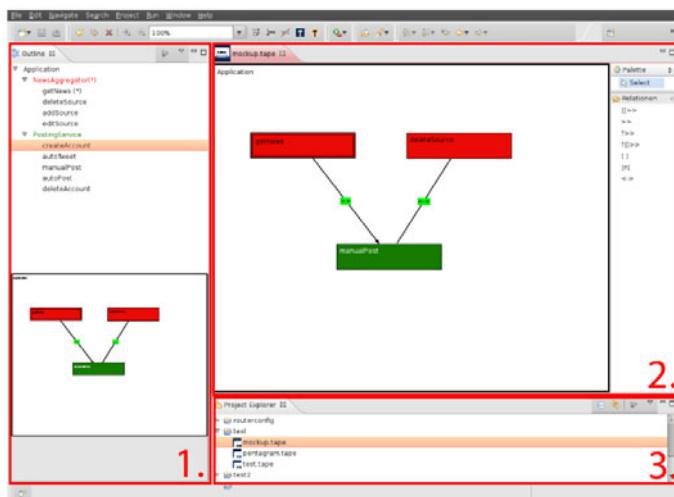


Fig. 1. Temporal Annotations Plugin for Eclipse (TAPE)

Our demonstration presents the graphical development process including the following steps to be accomplished by the user:

1. **Create a new TAPE project:** Initially, a WS has to be selected. During the modeling of the application's navigation and data flow further WS may be imported from the repository, which manages the URIs of available service descriptions. Subsequently, the initial WS operation has to be selected to specify the starting point of the application.
2. **Define the temporal relationships:** The application development starts by moving the initial operation into the *Editor*. It requires at least two operations to specify a binary annotation by using the annotation tool bar and clicking first on the left and then on the right operation. A feature for constraint checking can be activated to avoid incorrect model instances due to the fact that not all temporal annotations can be combined.
3. **Publish an application:** After modeling the whole application, the annotations can be published in the repository, which provides accessibility and

availability, and facilitates the exchange of created model instances. All annotation instances correspond to the extended Servface annotation model and may contain UI-related as well as application-specific annotations.

4. **Adapt an existing application:** To adapt an application to changing requirements, the published annotation file needs to be imported. Due to the fact that the tool does not store the layout information during the modeling process, we developed an algorithm that realizes the automatic and clearly arranged presentation of the WS operations and their temporal relationships.
5. **Generate an executable application:** By using transformation components, the application models can be transformed into executable applications for a variety of different platforms and devices.

Finally, we evaluated the scalability and usability of the authoring tool. The conducted user study with 10 participants has shown the development methodology is easily understood by developers and facilitates the development due to a clear layout and solely possesses small shortcomings with respect to the import and export of annotations.

3 Conclusion

In summary, the authoring tool demonstrates the feasibility of a hybrid light-weight development approach to create service-based interactive applications. The applied model relies solely on annotations to specify UI-related information and the navigation flow. We have shown that the hybrid approach provides the development of distinctive applications, which may differ in details with one single application model for a variety of platforms. As a next step, we plan to publish our extended Servface annotation model as well as the hybrid development methodology supporting platform-independency and variability. Future work includes the extension of the tool to cover all relevant aspects during development of service-based interactive applications and the development of code generators for mapping the modeled applications to different platforms. Based on this we intend to focus further on the results of the generation process.

References

1. Feldmann, M., Nestler, T., Muthmann, K., Jugel, U., Hubsch, G., Schill, A.: Overview of an End-user enabled Model-driven Development Approach for Interactive Applications based on Annotated Services. In: Proceedings of the 4th Workshop on Emerging Web Services Technology, pp. 19–28. ACM, New York (2009)
2. Paterno, F., Santoro, C., Spano, L.D.: Support for Authoring Service Front-Ends. In: Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 85–90. ACM, New York (2009)
3. Paterno, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: Proceedings of the 6th International Conference on Human-Computer Interaction, pp. 362–369. Chapman and Hall, Australia (1997)
4. Servface Consortium: Models for Service Annotations, User Interfaces and Service-based Interactive Applications, Deliverable 2.9 (2010)

A Comparative Evaluation of JavaScript Execution Behavior

Jan Kasper Martinsen¹, Håkan Grahn¹, and Anders Isberg²

¹ Blekinge Institute of Technology, Karlskrona, Sweden

{jan.kasper.martinsen,hakan.grahn}@bth.se

² Sony Ericsson Mobile Communications AB, Lund, Sweden

Anders.Isberg@sonyericsson.com

Abstract. JavaScript is a dynamically typed, object-based scripting language with runtime evaluation. It has emerged as an important language for client-side computation of web applications. Previous studies indicate some differences in execution behavior between established benchmarks and real-world web applications.

Our study extends previous studies by showing some consequences of these differences. We compare the execution behavior of four application classes, i.e., four JavaScript benchmark suites, the first pages of the Alexa top-100 web sites, 22 use cases for three social networks, and demo applications for the emerging HTML5 standard. Our results indicate that just-in-time compilation often increases the execution time for web applications, and that there are large differences in the execution behavior between benchmarks and web applications at the bytecode level.

1 Introduction

The World Wide Web is an important platform for many applications and application domains, e.g., social networking and electronic commerce. These type of applications are often referred to as web applications. Social networking web applications, such as Facebook, Twitter, and Blogger, have turned out to be popular, being in the top-25 web sites on the Alexa list [1]. All these three applications use JavaScript extensively. Further, we have found that 98 of the top-100 web sites use JavaScript to some extent.

JavaScript is a dynamically typed, object-based scripting language with runtime evaluation. The execution of a JavaScript program is done in a JavaScript engine [6], and several benchmarks have been proposed to evaluate its performance. However, previous studies show that the execution behavior differs between benchmarks and real-world web applications in several aspects [4,5].

We compare the execution behavior of four different application classes, i.e., (i) four established JavaScript benchmark suites, (ii) the start pages for 100 most visited web applications, (iii) 22 different use cases for popular social networks, and (iv) 109 demo applications for the emerging HTML5 standard. We extend previous studies with *three main contributions*: An extension of the execution behavior analysis with reproducible use cases of social network applications and

HTML5 applications, we show that just-in-time compilation often *increases* the execution time for web applications, and we provide a detailed instruction mix measurement and analysis. A more comprehensive set of results is found in [3].

2 Experimental Methodology

The experimental methodology is thoroughly described in [2]. We have selected a set of 4 application classes consisting of the first page of the 100 most popular web sites, 109 HTML5 demos from the JS1K competition, 22 use cases from three popular social networks (Facebook, Twitter, and Blogger), and a set of 4 benchmarks for measurements. We have measured and evaluated two aspects: the execution time with and without just-in-time compilation, and the bytecode instruction mix for different application classes. The measurements are made on modified versions of the GTK branch of WebKit (r69918) and Mozilla Firefox with the FireBug profiler.

Web applications are highly dynamic and the JavaScript code might change from time to time. We improve the reproducibility by modifying the test environment to download and re-execute the associated JavaScript locally (if possible). For each test an initial phase is performed 10 times to reduce the chances of execution of external JavaScript code.

Another challenge is the comparison between the social networking web applications and the benchmarks, since the web applications have no clear start and end state. To address this, we defined a set of use cases based on the behavior of friends and colleagues, and from this we created instrumented executions with the Autoit tool.

We modified our test environment in order to enable or disable just-in-time compilation. During the measurements, we executed each test case and application with just-in-time compilation disabled and enabled 10 times each, and selected the best one for comparison. We used the following relative execution time metric to compare the difference between just-in-time-compilation (JIT) and no-just-in-time-compilation (NOJIT):

$$T_{exe}(JIT)/T_{exe}(NOJIT) \geq 1$$

3 Experimental Results

3.1 Comparison of the Effect of Just-in-Time Compilation

In Figure 1 we present the relative execution time for the Alexa top-100 web sites, the first 109 JS1K demos, 24 SunSpider benchmarks, 6 Dromaeo benchmarks, and 10 JSBenchmarks. The results show that for 58 out of the top-100 web sites and for 50 out of 109 JS1K demos, JIT *increases* the execution time. When JIT fails, it increases the execution time by a factor of up to 75. In contrast, just-in-time compilation decreases the execution time for almost all benchmarks. In general, the penalty of a unsuccessful JIT compilation is larger in real-world web applications. However, the gain is also much larger for the largest decrease in execution time with a JIT compilation.

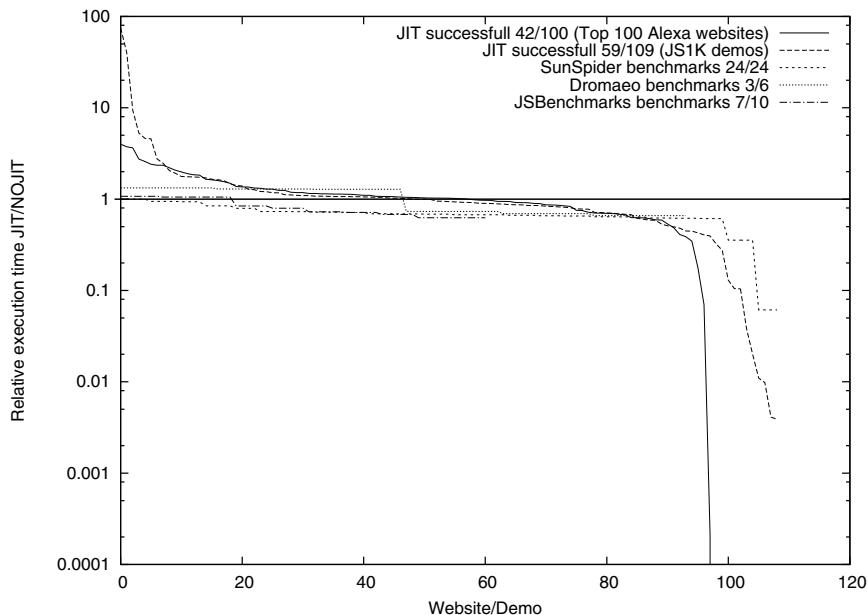


Fig. 1. Relative execution time $T_{exe}(JIT) / T_{exe}(NOJIT)$ for the top-100 Alexa web sites, the first 109 JS1K demos, 24 Sunspider benchmarks, 6 Dromaeo benchmarks, and 10 JSBenchmarks. A value larger than 1 means that JIT compilation *increases* the execution time.

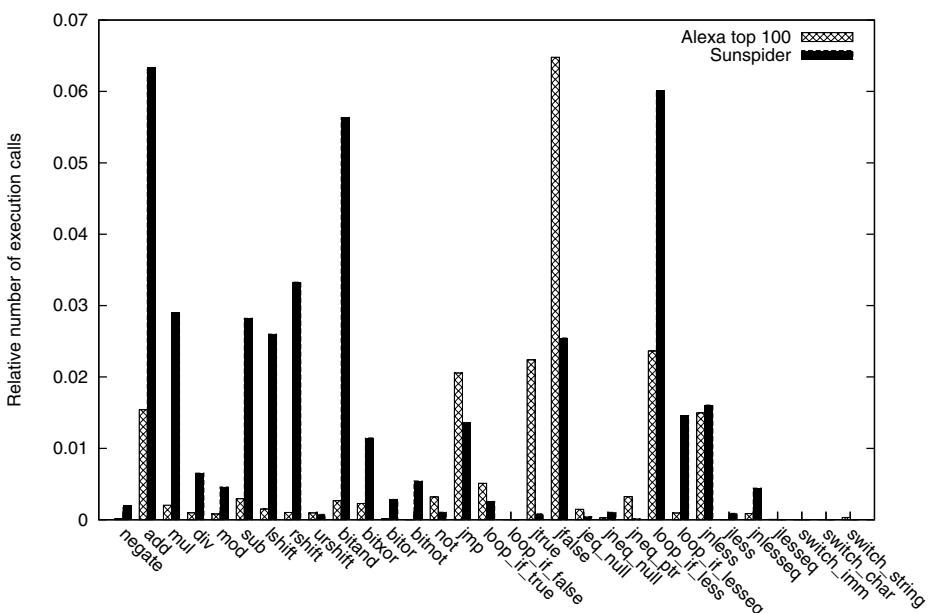


Fig. 2. Branch, jump, and arithmetic/logical related bytecode instructions for the Alexa top-100 web sites and the SunSpider benchmarks

3.2 Comparison of Bytecode Instruction Usage

We have measured the bytecode instruction mix for the selected benchmarks and for the Alexa top-100 sites. Figure 2 shows the results for the top-100 sites and the SunSpider benchmarks since they differ the most.

Our results show that the arithmetic/logical instructions and bit operations are used significantly more in the SunSpider benchmarks than in the web applications. We also find that general branch/jump instructions are more common in web applications, while loop instructions are more common in the benchmarks. The large number of `jmp` instructions indicates the importance of function calls in web applications.

4 Conclusions

Our most important results are that just-in-time compilation often *increases* the execution time of web applications and that the execution behavior of the benchmarks differs significantly from the web applications. The instruction mix gives an indication why loop-based optimization techniques often fails for web applications. These results call for alternative optimization techniques for web applications as well as benchmarks that better represents their workload.

Acknowledgments

This work was partly funded by the Industrial Excellence Center EASE - Embedded Applications Software Engineering, (<http://ease.cs.lth.se>).

References

1. Alexa: Top 500 sites on the web (2010), <http://www.alexa.com/topsites>
2. Martinsen, J.K., Grahn, H.: A methodology for evaluating JavaScript execution behavior in interactive web applications. In: Proc. of the 9th ACS/IEEE Int'l Conf. on Computer Systems And Applications, pp. XX–YY (December 2011)
3. Martinsen, J.K., Grahn, H., Isberg, A.: Evaluating four aspects of JavaScript execution behavior in benchmarks and web applications. Technical Report No. 2011:01, Blekinge Institute of Technology, Sweden (2011)
4. Ratanaworabhan, P., Livshits, B., Zorn, B.G.: JSMeter: Comparing the behavior of JavaScript benchmarks with real web applications. In: WebApps 2010: Proc. of the 2010 USENIX Conf. on Web Application Development, pp. 3–3 (2010)
5. Richards, G., Lebresne, S., Burg, B., Vitek, J.: An analysis of the dynamic behavior of JavaScript programs. In: PLDI 2010: Proc. of the 2010 ACM SIGPLAN Conf. on Programming Language Design and Implementation, pp. 1–12 (2010)
6. WebKit. The WebKit open source project (2010), <http://www.webkit.org/>

Designing a Step-by-Step User Interface for Finding Provenance Information over Linked Data

Enayat Rajabi¹ and Mohsen Kahani²

¹ Department of Engineering, Saveh Branch, Islamic Azad University, Saveh, Iran
erajab@iau-saveh.ac.ir

² Computer Engineering Department, Ferdowsi University Of Mashhad, Mashhad, Iran
kahani@um.ac.ir

Abstract. The proliferation of the use of Linked Data, and growth of Linked Open Data (LOD) cloud provide a good environment for interrelating previously isolated datasets. To encourage non-professional users to publish and find their required data easily, a good user interface is needed. Also, as users want to reach trustworthy or more up-to-date information in Linked Data, they would like to have access to the provenance data, as well. In this paper, a new method is presented that not only offers an easy interface for searching data in LOD cloud, but also provides provenance information of data.

Keywords: Linked Data, User-interface, Provenance, Metadata, LOD.

1 Introduction

With regarding to the growing availability of Linked Data on the Web [1], users are looking for approaches that do not limit them dealing with specific tasks.. They shouldn't be also limited to use specific data. Thus, the purpose of this paper is to help users to find their desired data and their provenance in an effective and easy way. Current Linked Data user interfaces such as search engines and browsers are discussed and analyzed in this research. In fact, currently, exploring the Linked Data cloud is difficult for users who are unfamiliar with semantic web concepts such as SPARQL and RDF [2]. Using easy graphical user interfaces for Linked Data, users can search effectively and find their exact data among Linked Datasets. Next, provenance information over Linked Data is considered and a metadata store is proposed for designing a step-by-step user interface and searching valid Linked Data. Finally, this approach is tested using a simple case study and simple metadata store.

2 Linked Data User-Interfaces

Linked Data user interfaces are various and can be classified into different kinds of user interfaces such as search engine, Linked Data browser, mash-ups, and triple query builder, etc. For example, Tabulator [3] browses RDF data on the Linked Data and uses an algorithm that traverses RDF data links and gives a dereferenceable URI. Fenfire [4] browser explores Linked Data with a graph view and highlights the use

cases of exploring and demonstrating Semantic Web data. DBpedia Faceted search¹ allows users to ask questions and then filters the results.

In most user interfaces for Linked Data, searching or exploring the data is performed similar to that of traditional search engines. Although it is possible to show users more accurate results, they should be able to search and filter the results again to reach the desired data. Users should be allowed to search easily and find the desired data without knowing the Linked Data principles. In some above mentioned user interfaces, users should be familiar with RDF triples. We found some types of interfaces such as Facets and query builders, as more effective ways for exploring data. Facets present filtered options in which end users can filter the results based on some related and limited fields. Because of using Facets and filtering the results in this kind of user interface, our approach is somehow similar to the Facets, but we use a Top-Down filtering method to reach the results, not filtering after showing the results (bottom-up filtering). It means that like advanced options in a search engine, users are guided to reach the results step-by-step by filtering datasets.

3 Finding Data Provenance

Veracity and quality of Linked Datasets are the most important things that should be determined in linked open data cloud regarding to openness of web data. Users can discern acceptance of the data by knowing about detailed history of them. Provenance information can be stored with other metadata or simply by itself [5]. In maintaining provenance, we should consider if it is immutable, or if it can be updated to reflect the current state of its ancestors. If provenance depends on users manually adding annotations instead of automatically collecting it, the burden on the user may prevent complete provenance from being recorded and available in a machine accessible form that has semantic value [6]. Users can also search for datasets based on their provenance metadata, such as locating all datasets that generated by executing a certain workflow. In order to use provenance, a system should allow rich and diverse means to access it.

4 The Proposed Approach

In the proposed approach, users can reach the desired data step-by-step from the metadata store to specific dataset(s) via search form as Figure 1 depicts. The metadata of datasets consist of dataset name, number of sub datasets, dataset ID, dataset usages, its domain and etc. Also details of dataset's relationships with other datasets, such as relation type can be stored in the data store. This helps us to know how each dataset is related with others and also helps users to find their desired data in an easy way.

An RDF data store is defined to store information about datasets and all datasets are encouraged to be registered in this metadata store. A registration process for checking the validity of data is required, as valid information about datasets should be presented to the users when they want to reach to the provenance information of data. After a while, a small cloud of LOD is created.

¹ <http://dbpedia.neofonie.de/browse>

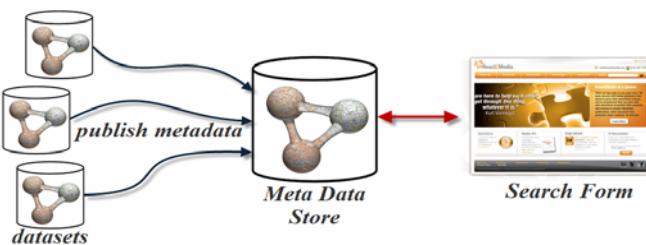


Fig. 1. Storing RDF Metadata in the Data Store

On the other hand, searching for content in a search engine basically comprises some steps like running a search, viewing the list of search results, selecting and viewing a document from the search results list and navigating the selected document. Each search engine usually has a search form that has a simple user interface. In a web search engine, one can restrict her searches by advanced options that are useful when she knows exactly what she wants. In the proposed approach, a search form in which users can select their keywords in a datasets list has been designed. The basic search is started by finding the keyword among the first level datasets in the metadata store in LOD. After finding the data, based on related datasets, users are guided with more information about it. Based on the selected keyword, the related datasets are retrieved from the data store. Each field in search form may depend on a selected field in the previous step in search page and the results of search will be filtered step by step.

Finding data provenance and confirming the validity of data in LOD cloud after finding the desired data is another advantage of having such a metadata store. Since the first level datasets and their relationships in RDF triple are stored, this method enables users to identify trustworthy of the provenance information via these relationships. Regarding different definitions which exist in RDFS and ontology (such as owl:same as and so on) they may have different routes and ways from the data provenance and the found data. Since we can keep relations of datasets in the metadata store, we can also use best path algorithms to find effective way to reach desired data. To maintain relationship between datasets, many other methods including parent-child relationship can be used.

5 A Case Study

In a case study, we created a simple metadata store (an RDF file) about sample datasets such as Geography, Databases, Search engines, etc. Then, we used SPARQL for fetching data from the file and Java and Jena² framework as programming language in IntelliJ IDEA³ environment. When a user select a dataset group, we list all types of datasets that are related to it, and then, in the next step, the user can select the desired dataset among the list shown (Figure 2).

² <http://jena.sourceforge.net/>

³ <http://www.jetbrains.com/idea/>



Fig. 2. A simple case study

6 Conclusion

In this paper, we considered a user interface of current Linked Data cloud and proposed a step-by-step user interface for easy and effective finding of data on LOD by designing a RDF metadata store for all datasets in LOD cloud. Having LOD metadata, users can be guided to find their data easily. On the other hand, when datasets metadata is created, valid data is delivered to users so that they can reach provenance information of datasets via the metadata. This is because it is possible to walk through datasets based on relationships defined in metadata store.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. International Journal on Semantic Web and Information Systems, Special Issue on Linked Data (2009) (in press)
2. Zembowicz, F., Opolon, D., Miles, S.: OpenChart: Charting Quantitative Properties in LOD. In: LDOW 2010 (2010)
3. Berners-Lee, et al.: Tabulator: Exploring and Analyzing Linked Data on the Semantic Web. In: Proceedings of the The 3rd International Semantic Web User Interaction Workshop (SWUI 2006) Workshop, Athens, Georgia (November 6, 2006)
4. Hastrup, T., Cyganiak, R., Bojars, U.: Browsing Linked Data with Fenfire. In: Hastrup, T., Cyganiak, R., Bojars, U. (eds.) Linked Data on the Web (LDOW 2008) Workshop, in Conjunction with WWW 2008 Conference (2008)
5. Margaritopoulos, T., Margaritopoulos, M., Mavridis, I., Manitsaris, A.: A Conceptual Framework for Metadata Quality Assessment. In: 2008 Proc. Int'l Conf. on Dublin Core and Metadata Applications, Berlin (2008)
6. Bose, R., Frew, J.: Composing Lineage Metadata with XML for Custom Satellite-Derived Data Products. In: SSDBM (2004)

Towards Behaviorally Enriched Semantic RESTful Interfaces Using OWL2

Irum Rauf and Ivan Porres

Åbo Akademi University, Dept. of Information Technologies, Turku, Finland
`{irauf, iporres}@abo.fi`

Abstract. In this paper, we discuss how to represent behavioral semantic RESTful interfaces using OWL2. The conceptual and behavioral model of the service interface are designed using UML and then given semantic representation in OWL2. These semantic RESTful interfaces carry information that can be used to validate web service and can also be published for automated discovery and composition processes. Different ontology reasoners can be used to validate the consistency of these semantic RESTful interfaces.

1 Introduction

Representational State Transfer (REST)[2] has become a popular approach for developing web services. Usually, RESTful services offer a simple interface to create, retrieve, update and delete (CRUD) resources. However, it is possible to create RESTful web services(WS) with complex operations beyond basic CRUD.

Designing and publishing such RESTful WS is not a trivial task. Previously, we highlighted this need and presented a design approach to create and publish behavioral RESTful WS interface [4]. The use of semantic technology with web services facilitates automation, discovery and composition of web services. We are interested in using semantic technology for representing behavioral RESTful WS interfaces that can be used with different ontology tools and offer automated solutions for RESTful interfaces.

In this work we use OWL2 to design behavioral semantic RESTful web service interface that contain application states. The service is represented as conceptual model(CM) and behavioral model(BM) using UML[6] and then semantically represented with OWL2 [1]. The semantic representation of RESTful WS interface can be validated for its consistency and satisfiability using ontology reasoners.

2 Behavioral RESTful Interfaces

We use a simple example of a RESTful hotel booking WS to demonstrate our work. The service takes payment from the customer and books a room in the hotel.

In our Hotel Booking (HB) RESTful example, we take *booking* as a central element since the service books a room. All other resources are linked to it and

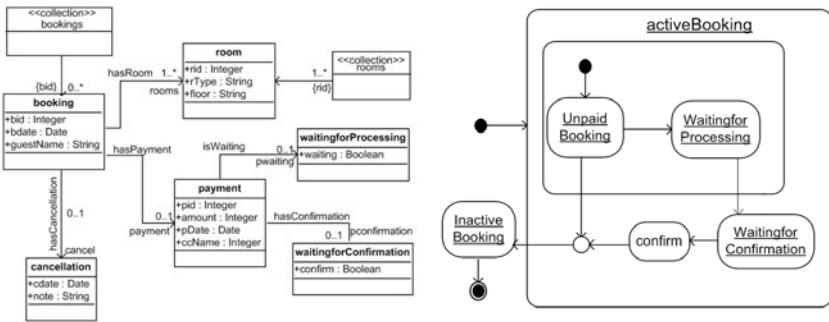


Fig. 1. (Left) CM for HB Web Service. (Right) BM of HB Web Service

are navigated through this central resource. The resources identified initially for HB service are shown in Figure 1(left). Figure 1(right) shows lifecycle of the *booking* resource with states representing different application states. A *booking* resource is created as an *activeBooking* and can have many other substates when it is active. A booking can be made inactive only if it is not waiting for the payment confirmation. Also, only an *inactiveBooking* can be deleted. For the detail implementation of the example, readers are referred to [5].

3 Behavioral RESTful Interfaces in OWL2

Each resource in CM is shown as a class in ontology. The ontology classes are connected via object properties that represent the association between resources, i.e. connectivity. Every class is declared disjoint with the other defining the fact that each class represents different objects. The listing below shows the excerpt of OWL 2 representation of CM in Figure 1(left). We do not semantically represent collection resources, class attribute and cardinalities as they would clutter interface ontology with details that are not involved in validating the interface. These details, however, can be added to make the semantic interface descriptive if required.

```

Declaration( Class ( booking ))
Declaration( Class ( room ))
Declaration( Class ( payment ))
...
Declaration( ObjectProperty ( containsBooking ))
Declaration( ObjectProperty ( hasRoom ))
Declaration( ObjectProperty ( hasCancellation ))
...
ObjectPropertyDomain( hasRoom booking )
ObjectPropertyRange( hasRoom room )
...
DisjointClasses( booking room payment cancellation
    waitPayment confirmPayment )

```

The BM of HB RESTful web service in Figure 1(right) results in emergence of many new concepts in our ontology. Each state represents a piece of information and can be exposed as a resource (ontology class).

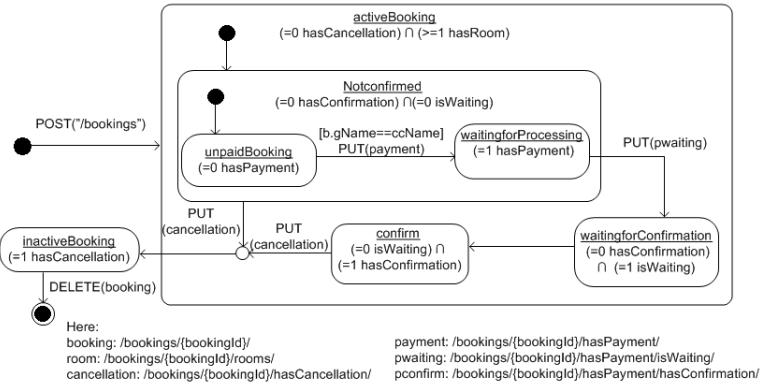


Fig. 2. Behavioral Model for HB RESTful WS with state invariants

```

Declaration( Class ( activeBooking ) )
Declaration( Class ( Notconfirmed ) )
Declaration( Class ( unpaidBooking ) )
.....

```

Next, the state hierarchy, i.e., class hierarchy of resources is defined that specify which resource is inherited from which resource. That is if state s_1 is a substate of s_2 , OWL2 class c_1 represents UML state s_1 and OWL2 class c_2 represents UML state s_2 then c_1 is a subclass of c_2 .

```

SubClassOf( unpaidBooking Notconfirmed )
SubClassOf( waitingforProcessing Notconfirmed )
....

```

The states at the same level of class hierarchy are mutually exclusive. The **DisjointClasses** axiom allows us to state this property of the state machine.

```

DisjointClasses( activeBooking inactiveBooking )
DisjointClasses( Notconfirmed confirm )
....

```

4 Behavioral Interfaces and Ontology Reasoners

The main design decision for BM is concerning the allowed methods. We allow four HTTP methods, i.e., GET, PUT, POST and DELETE, on the set of resources in state machine. GET is idempotent and is invoked to get the current state of the resource. PUT, POST and DELETE have side-effects and can trigger a transition from one application state to another. Figure 2 shows the behavioral model of our example annotated with service requests and state invariants. These state invariants link CM and BM of the interface. Each application state is defined by a state invariant, i.e., a boolean expression that links resources together to define an application state and is represented in OWL2 as:

```

SubClassOf (activeBooking
    ObjectIntersectionOf (ObjectExactCardinality(0
        hasCancellation cancel) ObjectExactCardinality(1 hasRoom
        room)))
SubClassOf (Notconfirmed
    ObjectIntersectionOf (ObjectExactCardinality(0 isWaiting
        waitPayment) ObjectExactCardinality(0 hasConfirmation
        confirmPayment)))
...

```

For detailed description on how state invariants are inferred for different types of states in UML protocol state machine, readers are referred to [3].

This OWL2 representation of state invariants provide semantic representation of applications states of a RESTful web service. Different OWL2 reasoners can be used to validate the consistency and satisfiability of these semantic behavioral interfaces. We must ensure that ontology describing CM should be consistent and all its resources should be satisfiable. Otherwise there cannot actually exist resources that conform to the interface described by our service. Each state invariant must be satisfiable and given two states that are at the same hierarchy level, their invariants should be mutually exclusive. For the classes to be satisfiable in the reasoner means that there exist resources that satisfy these application states.

5 Conclusion

In this paper, we discuss an approach to semantically represent behaviorally enriched RESTful WS interface using OWL2. These semantic behavioral RESTful interfaces can be published for automated discovery and composition and can also be used with ontology reasoners to discover inconsistencies at the design time.

References

1. Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., Smith, M.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. W3 Recommendation, (<http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/>)
2. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
3. Porres, I., Rauf, I.: From uml protocol statemachines to class contracts. In: Proceedings of the International Conference on Software Test, Verification and Validation (ICST 2010) (2010)
4. Porres, I., Rauf, I.: Modeling behavioral restful web service interfaces in uml. In: In the Proceedings of 26th Annual ACM Symposium on Applied Computing Track on Service Oriented Architectures and Programming (SAC 2011) (2011)
5. Rauf, I., Porres, I.: Beyond CRUD-REST: From Research to Practice, 1st edn. Springer, Heidelberg (2011)
6. OMG UML. 2.2 Superstructure Specification. OMG ed (2009), <http://www.omg.org/spec/UML/2.2/>

Taxonomy for Rich-User-Interface Components: Towards a Systematic Development of RIAs

Rosa Romero Gómez, David Díez Cebollero, Susana Montero Moreno,
Paloma Díaz Pérez, and Ignacio Aedo Cuevas

¹ DEI Lab - Computer Science Department
Universidad Carlos III de Madrid, Spain

david.diez@uc3m.es, {rmromero, smontero, pdp}@inf.uc3m.es,
aedo@ia.uc3m.es

Abstract. The development of Rich Internet Applications (RIAs) is based on the selection, assembly, and tailoring of Rich-User-Interface (RUI) components. While the user interface design is usually guided by principles, guidelines, and heuristics, there are not mechanisms for systematically selecting RUI components. Moreover, there is a lack of homogeneous classification criteria that hinders the selection of components and increases the relevance of experience designing web applications. To ease the search and the choosing of components by web-developers, this paper presents a taxonomy for classifying RUI components. The development of such a taxonomy has been based on both the study of relevant resources from the UI domain and the opinions of experts.

Keywords: Rich Internet Application, component libraries, taxonomy.

1 Introduction

Rich Internet Applications (RIAs) are ‘*web applications that offer the responsiveness, rich features, and functionality approaching of that desktop application*’ [2]. These capabilities are achieved by using Rich-User-Interface (RUI) components organized into libraries. These libraries – defined as ‘*reusable components collection for applications development*’ [1]– offer tens of components grouped by diverse criteria and related to different functionalities. Consequently, the search and choosing of the suitable component for a specific problem is not a trivial matter. The component selection process aims at determining which component, among those available, is the most suitable to fulfill a set of requirements of the UI design. The existence of classification schemes is therefore required to be more efficient when using component libraries [1] [3] [6], both during creation and selection of components. However, the experience in the use of RUI component libraries highlights the lack not only of homogeneous criteria for classifying RUI components but also of meaningful terminology. With the purpose of overcoming these limitations, we propose a taxonomy as such a classification artifact that allows categorizing components in a hierarchical way as a manner of easing the selection of RUI components.

This paper presents the taxonomy development process and the taxonomy itself based on both the study of relevant resources from UI domain and the opinions of

Subject-Matter Experts (SMEs hereafter). The rest of the paper is organized as follows. The taxonomy development process is described in section two; this process is divided into two levels – empirical and operational. As the result of such development process, the taxonomy is presented in section three. Finally, conclusions and intentions for further work are drawn in the last section.

2 The Taxonomy Development Process

The taxonomy development process refers to its definition by means of the execution of a well-known sequence of activities. In keeping with that, our process, based on the Nickerson's proposal [5], has been divided into two levels – empirical and operational– in order to provide an iterative process that guarantees the appropriate definition of the taxonomy. The empirical phase aims at identifying general characteristics of UI components in order to define a preliminary version of the taxonomy. To achieve this goal, the most direct factors contributing to the quality of the taxonomy [8] - such as corpus, the coverage of the source materials, and its structural model- has been considered. *Standards and guidelines, interaction design pattern libraries, and development technologies* have been selected because of their relevance and wide use for both UI development and UI design. Due to the disparity of characteristics possessed by complex RUI components – components that may be composed by other RUI components and provide more advanced functionalities, such as direct manipulation-, their categorization was the main challenge during the definition of this initial version of the taxonomy. Finally, this level ends with the definition of a draft taxonomy composed by four main categories – *Controls, Widgets, Containers, and Templates*- and a total of 81 subcategories.

The purpose of the operational phase is twofold. On the one hand, it is conceived to identify missing concepts, misunderstandings, or ambiguous terminology in order to refine our preliminary taxonomy. On the other hand, it allows us to validate the taxonomy and finalize the development process according to the agreement of experts over the relevance of the source materials, the definition of non-ambiguous and meaningful terms, and the consistency of hierarchical relationships throughout the taxonomy. Both activities are based on the feedback provided by SMEs in UI design and web development. Regarding the evaluation technique, we used a mix-questionnaire, i.e. the questionnaire combines both closed-ended questions and open-ended questions. Since *summarized rating scales* do not provide concrete values but categories, we used the *median* to identify the agreement of experts: the taxonomy will be considered as valid if the median of experts' opinion is equal or higher than four (agreement level). The operational phase compiles two iterations or evaluations carried out by eleven experts. The initial evaluation suggested that some categories of the taxonomy were ambiguous or misleading. Otherwise, the agreement of experts over the relevance of the corpus of the taxonomy, and the consistency of relationships throughout the taxonomy was validated. Finally, after the second round, the taxonomy has been considered as valid, due to the median of all indicators has achieved the agreement level.

3 Taxonomy for Rich-User-Interface Components

The final taxonomy consists of four main categories and a total of 89 subcategories. Its purpose is to classify RUI components in a hierarchical way in order to ease their search and choosing by web-developers. In keeping with this purpose, the definition of the taxonomy was based on intrinsic properties of the components and the common process of choosing components. On the one hand, we identified meta-characteristics that usually serve as basis for the search process: the *structure* – the provision of organization of both interface elements and contents and their relationships to each other- and the *behavior* – the interaction events available- of UI components. On the other hand, the systematic design of the UI [7] is supported by the order of the categories defined in this hierarchical-classification mechanism. The components compiled under *Containers* and *Controls* categories allow respectively establishing the main structure and the basic tasks carried out on the interface. The definition of the input/output information for the users is supported by the components classified under *Widgets* and *Interaction Design Patterns* category:

- **Containers:** it includes the RUI components used to organize the information. These components add support for *modality* – this property enables the developer to scope, or limit, a dialog box's modality blocking-, *drag and drop* – including moving, copying, or linking selected objects by dragging them from one location and dropping them over another- and default *look and feel characteristics*. Every component must be part of a containment *hierarchy* that has a top-level container as its root [4]. In keeping with that, containers have been divided into two subcategories: *top-level containers* that can hold other UI components; and *intermediate-level containers* that can hold and be held by other UI components.
- **Controls:** it includes the RUI components that allow users to carry out its tasks on the interface. Such category is divided into four subcategories: *imperative controls* used to initiate a function; *selection controls* used to select options or data; *entry controls* used to enter data; and *display controls* used to display the visual representation of information.
- **Widgets:** it includes the RUI components that provide a specific solution to a common design problem. These components use static, *predefined set of look and feel characteristics* in order to ease its configuration. Such category is divided into four subcategories: *entry widgets*; *selection widgets*; *display widgets*; and *navigation widgets*.
- **Interaction Design Patterns:** it includes the RUI components that provide a global solution to a common design problem. These components are usually constructed from aggregates of other RUI components and support setting the *input elements* according to each context of use. The subcategories defined are: *entry patterns*; *selection patterns*; *display patterns*; and *navigation patterns*.

4 Conclusions and Future Works

The existence of classification schemes is essential to be efficient when using component libraries. Nevertheless, existing RUI component libraries do not provide

homogeneous classification criteria, resulting in the need of high previous experience as RIA developer to search and choosing suitable components. With the purpose of overcoming these limitations, we have defined a taxonomy that allows us to categorize in a hierarchical way the structural and control elements of the UIs. This taxonomy has been considered as valid after the second round of evaluation according to the judgment of experts.

Further work will be guided to refine our taxonomy and prove its utility. On the one hand, we will carry out the empirical evaluation of both the quality of the taxonomy and its usefulness to select RUI components by means of the provision of a software tool. On the other hand, based on specific web-based interactive design scenarios, users testing will be carried out in order to assess the usefulness of the taxonomy to select RUI components.

Acknowledgements

This work has been partly supported by both the RIA – User Interface with Patterns and Components Reuse project funded by the Regional Government of Madrid (CAM) and the Universidad Carlos III de Madrid (UC3M) and the urTHEY project (TIN2009-09687) funded by the Ministry of Science and Innovation (MICINN) of the Government of Spain.

References

1. Curtis, N.: Modular Web Design: Creating Reusable Components for User Experience Design and Documentation. New Riders, Indianapolis (2009)
2. Deitel, P.J., Deitel, Deitel, H.M.: Ajax, Rich Internet Applications and Web Development for Programmers. Deitel Developers Series (2008)
3. Frakes, W.B., Pole, T.P.: An empirical study of representation methods for reusable software components. *IEEE Transactions on Software Engineering* 20(8) (1994)
4. Java Look and Feel Design Guidelines. Sun Microsystems Inc. Version 2.0 (2001), <http://java.sun.com/products/jlf/ed2/book/index.htm> (retrieved November 15, 2011)
5. Nickerson, R.C., Varshney, U., Muntermann, J., Isaac, H.: Taxonomy Development in Information Systems: Developing a Taxonomy of Mobile Applications. In: 17th European Conference on Information Systems (2009)
6. Pressman, R.S.: Software Engineering: A Practitioner's Approach. McGraw-Hill, New York (2005)
7. Stone, D., Jarrett, C., Woodroffe, M., Minocha, S.: User Interface Design and Evaluation (Interactive Technologies). Morgan Kauffman Series (2005)
8. Vogel, C., Powers, J.: Quality Metrics: How to Ensure Quality Taxonomies. *Information Today* (2000)

NAVTAG - A Network-Theoretic Framework to Assess and Improve the Navigability of Tagging Systems

Christoph Trattner

Graz Technical University of Technology
Knowledge Management Institute and
Institute for Information Systems and Computer Media
Inffeldgasse 21a/16c
A-8010 Graz
cstrattner@iicm.edu

Abstract. This paper presents NAVTAG – a network theoretical framework to assess and improve the navigability of tagging systems. The framework provides the developer of a tagging system with a simple to use and scalable tool to assess the navigability of a given tag network or a tag network that is generated by the NAVTAG framework using different tag cloud and resource list generation algorithms. To the best of our knowledge this framework is the first approach of a tool that is able to assess and improve the navigability of a given tagging system from a network-theoretic perspective.

Keywords: tagging systems, navigability, tag networks, network theory.

1 Introduction

Recently tagging systems [2] gained tremendously in popularity. While tagging systems were in the past typically associated with online bookmarking systems such as Delicious or CiteULike, the term is nowadays also connected with modern Web 2.0 web applications such as Amazon or LastFM. Basically, a tagging system is a tool that allows the user to apply light-weight key words - the so-called tags - to the resources of system. On LastFM for instance people apply tags to resources such as videos, photos or music. On Amazon as an other example of a modern tagging system people apply tags to product items such as books etc. Typically, developers provide the user with tagging functionality to allow the user to organize or describe the resources of a system. By visualizing the tags into the so-called tag clouds the users are then able to navigate to the resource of the system via tags.

While recent research has studied navigation in tagging systems from user interface [7], [8], [6] and information-theory [1] perspectives, the unique focus of our work is the network-theoretic analysis of tagging systems. In previous research it was observed that different tag cloud or resource list calculation algorithms [4],

[10], different tag taxonomy induction algorithms [3] or different types of tags [9] influence the navigability of a tagging system significantly. Basically, a navigable tagging system is defined as a system where the underlying tag network has a low diameter bounded by $\log(N)$, where N is the number of nodes in the network, and where the tag network has an existing giant component, i.e. a strongly connected component containing almost all resource of the tagging system [5]. An efficiently navigable tagging system is defined as a navigable network (see previous definition) and a network for which a decentralized searcher exists that is able to search the network in $\log(N)$ [5]. To measure these network properties a network-theoretic framework was developed. The framework is called NAVTAG – a network-theoretic framework to assess and improve the navigability of tagging systems. The novelty of the tool is on the one hand its uniqueness regarding the possibility to measure the navigability of tagging systems and on the other hand its uniqueness to do so on a network-theoretic level.

2 Approach and Implementation

Basically, the NAVTAG framework consist of three different modules: A tag network generation module, a tag-resource taxonomy generation module and a tag network analysis module. In the following sections the functionality and the implementation of the modules are described.

2.1 Tag Network Generation Module

The tag network generation module takes as input a given tagging dataset, a given tag cloud calculation algorithm and a given resource list generation algorithm and generates as output a tag network based on the given input tag dataset and algorithms. The module is implemented in Java and consists of the following three sub-modules:

Input-Reader: The input-reader basically stores the provided tagging data into a database module implemented with the Apache Lucene Search Engine.

Network Generator: This module generates the tag network. As input parameter the module takes a given tag cloud algorithm and/or a given resource list algorithm. The tag cloud algorithm implements the following interfaces routines $getTagCloud(r, n)$ where r is the currently processed resource and n the maximum tag cloud size and $getResourcList(r, t, k)$, where r is the currently processed resource, t the currently processed tag and k the maximum resource list size. Due to reasons of flexibility, generic interface routines are also provided.

Output-Writer: The output-writer module provides different output formats. Preferable, the module stores the generated tag network as bipartite or as tripartite tag network into a file. Additionally, the module outputs the resource-resource network and a tag-tag network of the tag network as well.

2.2 Tag-Resource Taxonomy Generation Module

Additionally to the network generation module NAVTAG provides a tag-resource taxonomy generation module. The module allows to generate different tag or resource taxonomies based on the tagging dataset provided by tag network generation module. Since tag-resource taxonomies are a popular approach to improve the navigability of tagging systems [4], the module implements popular tag taxonomy induction algorithms such as Heymann, Deg/Cos and Deg/Cooc [4]. As well as the tag network generation module, this module is implemented in Java.

2.3 Tag Network Analysis Module

The tag network analysis module is an extension of the Stanford *SNAP*¹ network library. Due to reasons of performance and scalability, this module is implemented in C++. By default the module provides the functionality to measure network properties such as in- and out-degree of a given network as well as navigability properties such as the size of largest strongly connected component or the efficient diameter of given network. To measure the efficiency of a tag network, a hierarchical decentralized searcher was implemented [3]. The searcher is able to measure whether a tag network is navigable in $\log(N)$ or not, i.e. the searcher measures if the given tag network is also efficiently navigable.

As output the module generates a “navigability”-statistic file containing detailed information about the size of the largest strongly connected component, the efficient diameter of the tag network and the average number of hops of the searcher to reach a given number of nodes pairs in the tag network. The results are printed to a file in plain text format. Additional figure representations of the results in PS format are produced.

3 Results

All in all, the NAVTAG framework was successfully deployed in parts in a number of projects [4] [10], [3], [9] related to the field of tagging systems. For instance in [3], [4] the framework was successfully used to assess the navigability of different tagging networks (billion order) such BibSonomy, CiteULike, Delicious, LastFM or Flickr. The memory consumed by NAVTAG in the case of Delicious, the largest tag network on the web today was less than one GB. In [9] the framework was used to investigate and develop novel tag cloud and resource list generation algorithms. In [9] the framework was used to investigate the navigational utility of the so-called Google query tags compared to tags generated by users. In our latest research regarding the navigability of tagging systems the framework was deployed to assess the navigational utility of different tag taxonomy induction algorithms [3].

¹ <http://snap.stanford.edu/>

4 Conclusions and Future Work

In this work, NAVTAG - A network theoretical framework to assess and improve the navigability of tagging systems was presented. The framework provides the developer of a tagging system with a simple to use and scalable tool to assess the navigability of a given tag network or a tag network that is generated by the NAVTAG framework using different tag cloud and resource list generation algorithms. The framework was successfully deployed for a couple of major projects related to the field of tag navigation. To the best of our knowledge this framework is the first approach of tool that is able to assess and improve the navigability of a given tagging system. Future work will include further improvements towards the functionality of the framework and the release through the Google code project service.

Acknowledgments. This work is funded by - *BMVIT - the Federal Ministry for Transport, Innovation and Technology*, program line *Forschung, Innovation und Technologie für Informationstechnologie*, project *NAVTAG – Improving the navigability of tagging systems*.

References

1. Chi, E.H., Mytkowicz, T.: Understanding the efficiency of social tagging systems using information theory. In: HT 2008: Proc. of the Nineteenth ACM Conference on Hypertext and Hypermedia, pp. 81–88. ACM, New York (2008)
2. Hammond, T., Hannay, T., Lund, B., Scott, J.: Social bookmarking tools (i): A general review. D-Lib Magazine 11(4) (2005)
3. Helic, D., Strohmaier, M., Trattner, C., Muhr, M., Lerman, K.: Pragmatic evaluation of folksonomies. In: Proceedings of the 20th International Conference on World Wide Web, WWW 2011, pp. 417–426. ACM, New York (2011)
4. Helic, D., Trattner, C., Strohmaier, M., Andrews, K.: Are tag clouds useful for navigation? a network-theoretic analysis. International Journal of Social Computing and Cyber-Physical Systems (2011)
5. Kleinberg, J.M.: Small-world phenomena and the dynamics of information. In: Advances in Neural Information Processing Systems (NIPS), vol. 14. MIT Press, Cambridge (2001)
6. Mesnage, C.S., Carman, M.J.: Tag navigation. In: SoSEA 2009: Proc. of the 2nd International Workshop on Social Software Engineering and Applications, pp. 29–32. ACM, New York (2009)
7. Rivadeneira, A.W., Gruen, D.M., Muller, M.J., Millen, D.R.: Getting our head in the clouds: toward evaluation studies of tagclouds. In: Proc. of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2007, pp. 995–998. ACM, New York (2007)
8. Sinclair, J., Cardew-Hall, M.: The folksonomy tag cloud: when is it useful? Journal of Information Science 34, 15 (2008)
9. Trattner, C., Helic, D.: Linking related documents: Combining tag clouds and search queries. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 486–489. Springer, Heidelberg (2010)
10. Trattner, C., Helic, D., Strohmaier, M.: On the construction of efficiently navigable tag clouds using knowledge from structured web content. J-jucs 17(4), 565–582 (2011)

Author Index

- Abel, Fabian 28, 167
Abid, Adnan 44
Akkermans, Vincent 289
Alhosban, Amal 59
Araújo, Samur 28

Baez, Marcos 361
Barbagallo, Donato 152
Barbieri, Davide Francesco 363
Benjamin, Kamara 74
Bislimovska, Bojana 90
Blomme, Dieter 375
Bochmann, Gregor von 74
Bohøj, Morten 105
Bomfim, Mauricio Henrique de Souza 121
Boor, Aart-Jan 274
Bouvin, Niels Olof 105
Bozzon, Alessandro 1, 90, 363, 367, 371
Brambilla, Marco 1, 90, 363, 367, 371
Budiselic, Ivan 346

Cappiello, Cinzia 137, 152
Carlson, Jan 228
Casati, Fabio 361
Celik, Ilknur 167
Ceri, Stefano 1, 363
Chaisatien, Prach 182
Cigardi, Laura 371
Cioria, Luca 367
Comai, Sara 371
Conlan, Owen 391
Cuevas, Ignacio Aedo 411

Daniel, Florian 137
Delac, Goran 346
Desruelle, Heiko 375
Díez Cebollero, David 411
Díaz Pérez, Paloma 411
Dincturk, Mustafa Emre 74

Faliagka, Evanthia 379
Feldmann, Marius 395
Firmenich, Sergio 198
Francalanci, Chiara 152

Frasincar, Flavius 274
Fraternali, Piero 90, 367
Furche, Tim 13

Gaedke, Martin 387
Gammelmark, Henrik 105
Gao, Qi 28
Gielen, Frank 375
Gordillo, Silvia 198
Gottlob, Georg 13
Grahn, Håkan 399
Granić, Andrina 383
Guo, Xiaonan 13

Hashmi, Khayyam 59
Heinrich, Matthias 387
Herder, Eelco 258
Hogenboom, Frederik 274
Houben, Geert-Jan 28, 167

Isberg, Anders 399
Ishikawa, Hiroshi 331

Jourdan, Guy-Vincent 74

Kahani, Mohsen 403
Katz, Philipp 395
Kawase, Ricardo 258
Koidl, Kevin 391
Koschmider, Agnes 137
Kozanidis, Lefteris 379

Lew, Philip 214
Liebing, Christian 395

Malik, Zaki 59
Marangunić, Nikola 383
Maras, Josip 228
Marchese, Maurizio 361
Martinsen, Jan Kasper 399
Matera, Maristella 137, 152, 367
Medjahed, Brahim 59
Mitrović, Ivica 383
Montero Moreno, Susana 411
Mosig, Jan 395

- Nebeling, Michael 243
Niederée, Claudia 258
Norrie, Moira C. 243
- Olsina, Luis 214
Onut, Iosif Viorel 74
- Papadakis, George 258
Pasini, Chiara 363
Picozzi, Matteo 137, 152
Porres, Ivan 407
Prutsachainimmit, Korawit 182
- Radelaar, Joni 274
Rahm, Erhard 304
Rajabi, Enayat 403
Rauf, Irum 407
Restagno, Luca 289
Rizzo, Giuseppe 289
Romero Gómez, Rosa 411
Rossi, Gustavo 198
- Schallhart, Christian 13
Schill, Alexander 395
Schwabe, Daniel 121
Sellers, Andrew 13
Servetti, Antonio 289
- Sprega, Gabriele 152
Stamou, Sofia 379
Štula, Maja 228
- Tagliasacchi, Marco 44
Tettamanti, Luca 363
Thor, Andreas 304
Tokuda, Takehiro 182
Trattner, Christoph 415
Tsakalidis, Athanasios 379
Tzimas, Giannis 379
- Vadacca, Salvatore 363
van Dam, Jan-Willem 274
Vandic, Damir 274
Volonterio, Riccardo 363
- Wade, Vincent 391
Wang, Cheng 13
Winckler, Marco 198
- Yahyaoui, Hamdi 319
Yokoyama, Shohei 331
- Zagorac, Srđan 363
Zhioua, Sami 319
Zuzak, Ivan 346