

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*University of Dortmund, Germany*

Madhu Sudan

*Massachusetts Institute of Technology, MA, USA*

Demetri Terzopoulos

*New York University, NY, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Moshe Y. Vardi

*Rice University, Houston, TX, USA*

Gerhard Weikum

*Max-Planck Institute of Computer Science, Saarbruecken, Germany*

**Springer**

*Berlin*

*Heidelberg*

*New York*

*Hong Kong*

*London*

*Milan*

*Paris*

*Tokyo*

C.V. Ramamoorthy Roger Y. Lee  
Kyung Whan Lee (Eds.)

# Software Engineering Research and Applications

First International Conference, SERA 2003  
San Francisco, CA, USA, June 25-27, 2003  
Selected Revised Papers



Springer

Volume Editors

C.V. Ramamoorthy

University of California at Berkeley, Computer Science Department  
Berkeley, CA 94720-1776, USA  
E-mail: ram@cs.berkeley.edu

Roger Lee

Central Michigan University  
Software Engineering & Information Technology Institute  
Mount Pleasant, MI 48859, USA  
E-mail: lee@cps.cmich.edu

Kyung Whan Lee

Chung-Ang University, School of Computer Science and Engineering  
Seoul 156-756, Korea  
E-mail: kwlee@object.cau.ac.kr

Library of Congress Control Number: 2004104592

CR Subject Classification (1998): D.2, I.2.11, C.2.4, H.5.2-3, K.6

ISSN 0302-9743

ISBN 3-540-21975-7 Springer-Verlag Berlin Heidelberg New York

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

Springer-Verlag is a part of Springer Science+Business Media

[springeronline.com](http://springeronline.com)

© Springer-Verlag Berlin Heidelberg 2004  
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik  
Printed on acid-free paper SPIN: 10999295 06/3142 5 4 3 2 1 0

## **Foreword**

It was our great pleasure to extend a welcome to all who participated in SERA 2003, the first world-class International Conference on Software Engineering Research and Applications, which was held at Crowne Plaza Union Square Hotel, San Francisco, California, USA. The conference was sponsored by the International Association for Computer and Information Science (ACIS), in cooperation with the Software Engineering and Information Technology Institute at Central Michigan University.

This conference was aimed at discussing the wide range of problems encountered in present and future high technologies. In this conference, we had keynote speeches by Dr. Barry Boehm and Dr. C.V. Ramamoorthy and invited talks by Dr. Raymond Yeh, Dr. Raymond Paul, Dr. Mehmet Şahinoglu, which were fruitful to all who participated in SERA 2003.

We would like to thank the publicity chairs and the members of our program committees for their work on this conference. We hope that SERA 2003 was enjoyable for all participants.

C.V. Ramamoorthy

## Preface

It was indeed our pleasure to welcome all of you to the 1st ACIS International Conference on Software Engineering Research and Applications (SERA 2003), June 25–27, 2003, Crowne Plaza Union Square Hotel, San Francisco, California, USA. SERA 2003 featured excellent theoretical and practical contents contributed by the international community in the areas of formal methods and tools, requirements engineering, software process models, communication systems and networks, software quality and evaluation, software engineering, networks and mobile computing, parallel/distributed computing, component-based software development, artificial intelligence and applications, software testing, reuse and metrics, database retrieval, HCI, software standards, computer security, software architectures, and modeling.

We received 104 papers (our Program Co-chair Prof. Kyung Whan Lee received 45 papers and Prof. Roger Lee received 59 papers) from 22 different countries around the world for possible presentation at the conference. Among those papers, a large number were high-quality paper submissions. Three reviewers reviewed each paper. After the completion of the peer-review process, our International Program Committee selected 53 papers, which gave about a 49% acceptance rate for publication in the ACIS conference proceedings. Of the 53 presented papers at the conference our international program committee selected 23 outstanding papers for publication in Springer's LNCS as the post-conference proceedings.

We would like to express our sincere appreciation to the following people for contributing to the success of this conference: Dr. C.V. Ramamoorthy, conference chair; Dr. Barry Boehm and Dr. C.V. Ramamoorthy, our keynote speakers; Dr. Raymond Yeh, Dr. Raymond Paul and Dr. Mehmet Şahinoglu, our invited speakers; the publicity chairs and the International Program Committee members who rose beyond the call of duty, contributing a large percentage of their time; the referees who accepted the last-minute requests for extra reviews; the session chairs who presided over the sessions; all the authors, attendees, and presenters who really made this conference possible and successful; Prof. Dr. Walter Dosch for his extra time and effort in communicating with Springer-Verlag about LNCS post-conference proceedings; and, finally, the editorial staff at Springer-Verlag. Furthermore, we would like to extend special thanks to Alfred Hofmann of Springer-Verlag who made this publication possible.

Roger Y. Lee  
Kyung Whan Lee

# **Organizing Committee**

## **Conference Chair**

Prof. C.V. Ramamoorthy, Computer Science Department  
University of California, Berkeley

## **Program Co-chairs**

Prof. Roger Lee  
Computer Science Department  
Central Michigan University (USA)

Prof. Kyung Whan Lee  
Computer Engineering Department  
Chung-Ang Univ. (Seoul, Korea)

## **Publicity Co-chairs**

**Australia**  
Dr. Sebastian Ng  
Swinburne Univ. of Technology

**The Baltic States & Scandinavia**  
Prof. Dr. Juri Vain  
Technical Univ. of Tallinn

**Canada/Africa**  
Prof. Sylvanus A. Ehikoya  
Univ. of Manitoba

**China**  
Dr. Liu Jingnam  
Wuhan University

**Germany**  
Prof. Dr. Walter Dosch  
Univ. of Lubeck

**UK**  
Prof. Paul Garratt  
Univ. of Southampton

**Hong Kong**  
Dr. Pak-Lok Poon  
Hong Kong Polytechnic Univ.

**Italy**  
Prof. Susanna Pelagatti  
Univ. of Pisa

**Japan**  
Dr. Yoshihiro Akiyama  
IBM Global Services-Japan

**Korea**  
Prof. Haeng-Kon Kim  
Catholic Univ. of Daegu

**Lebanon**  
Prof. Ramzi Haraty  
Lebanese American Univ.

**Russia**  
Prof. Alexandre Zamulin  
Institute for Informatics Systems,  
Novosibirsk

**South America**  
Prof. Silvia T. Acuna  
Uni. Naci. de Santiago-Argentina

**USA**  
Prof. Chia-Chu Chiang  
Univ. of Arkansas-Little Rock

## International Program Committee

Doo Hwan Bae  
(KAIST, Korea)

Jongmoon Baik  
(Motorola Labs, USA)

Chia-Chu Chiang  
(Univ. of Arkansas-Little Rock, USA)

Byungju Choi  
(Ewha Womans Univ., Korea)

Walter Dosch  
(Univ. of Lubeck, Germany)

Sylvanus A. Ehikioya  
(Univ. of Manitoba, Canada)

Paul Garratt  
(Univ. of Southampton, UK)

Gongzhu Hu  
(Central Michigan Univ., USA)

Ha Jin Hwang  
(Catholic Univ. of Daegu, Korea)

Naohiro Ishii  
(Nagoya Institute of Technology, Japan)

Matjaz Juric  
(Univ. of Maribor, Slovenia)

Yanggon Kim  
(Towson Univ., USA)

Oh Hyun Kwon  
(Tongmyung Univ. of IT, Korea)

EunSeok Lee  
(Sungkyunkwan Univ., Korea)

Juhnyoung Lee  
(IBM T.J. Watson Research Center, USA)

Xiaoqing Frank Liu  
(Univ. of Missouri-Rolla, USA)

Brian Malloy  
(Clemson University, USA)

Jun Park  
(Samsung, Korea)

David Primeaux  
(Virginia Commonwealth Univ., USA)

Yeong Tae Song  
(Towson Univ., USA)

Atsushi Togashi  
(Shizuoka University, Japan)

Yoshiichi Tokuda  
(SONY Corporation, Japan)

Juan Trujillo  
(Univ. of Alicante, Spain)

Cui Zhang  
(Univ. of California-Sacramento, USA)

## Additional Reviewers

Jiri Adamek (Charles University, Czech Republic)  
Thomas Ahlswede (Central Michigan University, USA)  
Gail Ahn (Univ. of North Carolina-Charlotte, USA)  
Carsten Albrecht (University of Lubeck, Germany)  
Thomas Bures (Charles University, Czech Republic)  
Jie Dai (Central Michigan University, USA)  
Amir Madany Mamlouk (University of Lubeck, Germany)  
Miho Osaki (Shizuoka University, Japan)  
Petr Planuska (Charles University, Czech Republic)  
Gilles Roussel (Université de Marne-la-Vallée, France)  
Remzi Seker (Embry-Riddle Aeronautical University, USA)  
Chong-wei Xu (Kennesaw State University, USA)

# Table of Contents

## Keynote Addresses

- Balancing Agility and Discipline: A Guide for the Perplexed ..... 1  
Barry Boehm

- A Study of Feedback in Software Supported Networked Systems ..... 2  
C.V. Ramamoorthy

## Invited Talks

- Why Great Organizations Are Great? *The Art of Business* ..... 3  
Raymond T. Yeh

- Future of Computer Software Systems: Commodity or Service? ..... 4  
Raymond Paul

- An Exact Reliability Block Diagram Calculation Tool  
to Design Very Complex Systems ..... 5  
Mehmet Şahinoglu

## Formal Methods

- Computer-Aided Refinement of Data Structures  
on Higher-Order Algebraic Specifications ..... 7  
Walter Dosch and Sönke Magnussen

- Analyzing Relationships to the Quality Level  
between CMM and Six Sigma ..... 34  
Ji-Hyub Park, Ki-Won Song, Kyung Whan Lee, and Sun-Myung Hwang

- Deterministic End-to-End Guarantees for Real-Time Applications  
in a DiffServ-MPLS Domain ..... 51  
Steven Martin, Pascale Minet, and Laurent George

## Component-Based Software Engineering

- Measuring and Communicating Component Reliability ..... 74  
John D. McGregor, Judith A. Stafford, and Il-Hyung Cho

- Toward Component-Based System: Using Static Metrics and Relationships  
in Object-Oriented System ..... 87  
Eunjoo Lee, Byungeong Lee, Woochang Shin, and Chisu Wu

Communication Style Driven Connector Configurations . . . . .	102
---	-----

*Tomas Bures and Frantisek Plasil*

A Study on the Specification for e-Business Agent Oriented Component Based Development . . . . .	117
---	-----

*Haeng-Kon Kim, Hae-Sool Yang, and Roger Y. Lee*

## Software Process Models

Towards Efficient Management of Changes in XML-Based Software Documents . . . . .	136
--	-----

*Geunduk Park, Woochang Shin, Kapsoo Kim, and Chisu Wu*

Model-Based Project Process Analysis Using Project Tracking Data . . . . .	148
--	-----

*Kyung-A Yoon, Sang-Yoon Min, and Doo-Hwan Bae*

## Software Testing and Quality Issues

A COM-Based Customization Testing Technique . . . . .	168
---	-----

*Hoijin Yoon, Eunhee Kim, and Byoungju Choi*

Design Defect Trigger for Software Process Improvement . . . . .	185
--	-----

*EunSer Lee, Kyung Whan Lee, and Keun Lee*

Certification of Software Packages Using Hierarchical Classification . . . . .	209
--	-----

*Jaewon Oh, Dongchul Park, Byungejung Lee, Jongwon Lee,  
Euyseok Hong, and Chisu Wu*

An Automatic Test Data Generation System Based on the Integrated Classification-Tree Methodology . . . . .	225
---	-----

*Andrew Cain, Tsong Yueh Chen, Doug Grant, Pak-Lok Poon,  
Sau-Fun Tang, and T.H. Tse*

## Requirements Engineering, Reengineering, and Performance Analysis

Normalizing Class Hierarchies Based on the Formal Concept Analysis . . . . .	239
--	-----

*Suk-Hyung Hwang, Sung-Hee Choi, and Hae-Sool Yang*

QoS-Guaranteed DiffServ-Aware-MPLS Traffic Engineering with Controlled Bandwidth Borrowing . . . . .	253
---	-----

*Youngtak Kim and Chul Kim*

Architecture Based Software Reengineering Approach for Transforming from Legacy System to Component Based System through Applying Design Patterns . . . . .	266
---	-----

*Jung-Eun Cha, Chul-Hong Kim, and Young-Jong Yang*

**Knowledge Discovery and Artificial Intelligence**

Increasing the Efficiency of Cooperation among Agents by Sharing Actions .....	279
<i>Kazunori Iwata, Mayumi Miyazaki, Nobuhiro Ito, and Naohiro Ishii</i>	
Applying Model Checking Techniques to Game Solving.....	290
<i>Gihwon Kwon</i>	
A Fuzzy Logic-Based Location Management Method for Mobile Networks .....	304
<i>Ihn-Han Bae, Sun-Jin Oh, and Stephan Olariu</i>	
A Next Generation Intelligent Mobile Commerce System .....	320
<i>Eunseok Lee and Jionghua Jin</i>	

**Database Retrieval and Human Computer Interaction**

Indexing XML Data for Path Expression Queries .....	332
<i>Gongzhu Hu and Chunxia Tang</i>	
Improving Software Engineering Practice with HCI Aspects .....	349
<i>Xavier Ferre, Natalia Juristo, and Ana M. Moreno</i>	

**Software Security**

Reliability Assurance in Development Process for TOE on the Common Criteria .....	364
<i>Haeng-Kon Kim, Tai-Hoon Kim, and Jae-Sung Kim</i>	

<b>Author Index</b> .....	377
---------------------------	-----

## **Keynote Address**

# **Balancing Agility and Discipline: A Guide for the Perplexed**

Barry Boehm

Director, USC Center for Software Engineering  
TRW Professor of Software Engineering  
Computer Science Department  
University of Southern California  
Los Angeles, CA 90089

Future software applications increasingly need the agility to adapt to rapid change and the discipline to provide dependable software-intensive products and services. Considerable perplexity currently reigns in assessing the ability of agile methods such as Extreme Programming, Scrum, Crystal, and Adaptive Software Development in providing both agility and discipline, in comparison with the use of plan-driven methods oriented around the use of frameworks such as the software Capability Maturity Model (CMM) and the CMMI-Integrated (CMMI).

This presentation, based on a forthcoming Addison Wesley book with Richard Turner, "Balancing Agility and Discipline: A Guide for the Perplexed," will examine the sources of this perplexity, and will assess the relative "home grounds" of agile and plan-driven methods: the range of software applications, management, technical, and personnel characteristics for which the two classes of methods work best. It will also summarize two case studies showing how a project can succeed by starting from an agile or disciplined method and adding aspects of the complementary method.

The presentation will then provide five scales which a project or organization can use to determine how well it fits the use of agile or disciplined methods: size, criticality, personnel capability, dynamism, and organizational culture. It will provide and illustrate by example projects a risk-based approach by which a project can use its risk patterns to tailor an appropriate mix of agile and plan-driven methods to cope with the risks. It will conclude by summarizing an approach by which organizations can assess their current and future needs for agility and discipline, and can develop a cost-effective strategy for adapting their organizations to meet their particular combination of needs for agility and discipline.

## **Keynote Address**

# **A Study of Feedback in Software Supported Networked Systems**

C.V. Ramamoorthy

Professor Emeritus, University of California, Berkeley  
Sr. Research Fellow, ICC Institute, The University of Texas, Austin  
ram@cs.berkeley.edu

The purpose of this paper is to study the concept of feedback in software supported networked devices, appliances and systems. These systems use system generated feedback signals (requests) to obtain help from external service providers/vendors during system/application emergencies and other contingencies. Similar techniques are also used to keep external entities apprised of the system's progress and health. Future systems, including appliances, devices and complex entities operated by humans would be networked and would be allowed to interact amongst themselves as well as with service providers under well-defined specified but constrained conditions. We explore many advantages and opportunities obtainable through the use of external services triggered by feedback from the system. These services include real time maintenance, troubleshooting and diagnosing the causes of failures, recovery from failures, operational support and help, protection against security and safety infringements, congestion and conflict resolution, overload help through resource sharing etc. Currently in certain network products, a technique of reverse feedback (from the service provider to the system) is in vogue. Some networked products, namely those by Microsoft and IBM, request and receive periodic updates and revisions directly from their vendors. These functions provide the customer with better quality of service and extend the longevity of the system. These also bring the consumers closer to the product developers by suggesting future needs, opinions, improvements and refinements.

In this presentation we generalize the concept of feedback generation and processing in a networked environment and study many of its ramifications and issues. We categorize the feedback types with examples. We trace the life cycle of a feedback-generated request from the time of its creation to the point where the request is consummated, exploring and exposing the varieties of issues on the way.

We show such systems emphasize continued automation of services, which can replace many user-operated chores. In particular, we discuss parallel multiple diagnosis, the concept of checking the checkers, resource sharing by reconfiguration in overcapacity-systems, or via grid or cluster protocols, replacing a failing executing environment with a virtual healthy replacement, etc. We develop some models and methods of web-based service support. We describe and explore techniques such as non-intrusive shadow and standby programming and porous programming and their applications to reliable and secure feedback supported systems. We conclude by identifying challenging problems and research issues.

## **Invited Talk**

# **Why Great Organizations Are Great?**

## ***The Art of Business***

Raymond T. Yeh

Senior Research Fellow  
ICC Institute  
The University of Texas  
Austin, Texas

This presentation will share a study, being documented in a forthcoming book of the same title that answers this question. Our research discovered that a Great organization must first of all have a *soul*, which creates meaning for people working there. Such an organization knows where it is going and somehow always seems to *flow* with the changing world outside to get there in time. A Great organization smartly *leverages* on its environment, including its competitors, to make sure that its resources are effectively and efficiently utilized. It is also the *master* of its trade by being consistently on the edge while maintaining balances. Finally, a Great organization is made of *leaders* who help to actualize the organization's vision by aligning their own dreams to it.

Great organizations studied here spread around the globe including for-profit, non-profit, mature and young organizations. The practice of these strategic arts will help a company to charter a steady course in the quest for its vision and can resist temptations/pressures from its environment. For example, our society's focus on short term bottom-line many times causes a company to lose its *soul* as evidenced by Enron, WorldCom, Andersen Consulting, etc. This talk will also explore insights of great leaders in the application of these arts to guide their organizations to unprecedented successes.

## **Invited Talk**

# **Future of Computer Software Systems: Commodity or Service?**

Raymond Paul

Department of Defense  
Washington, DC

Web Services (WS) received significant attention recently by government agencies and computer industries. WS provides a new architecture/paradigm for building distributed computing applications based on XML. It provides a uniform and widely accessible interface to glue the services implemented by the other middleware platform over the Internet by utilizing standard Internet protocols such as WSDL, SOAP and UDDI.

While WS is still early in its maturing processes, many issues still need to be addressed, e.g., finalizing draft specifications, runtime verification and validation, and quality assurance by the UDDI servers, many keen observers agree that WS represent a new significant trend for software systems integration that will be developed, structured, acquired, and maintained. For example, instead of buying and maintaining software, software can be leased and downloaded when needed. Thus, in this way, software upgrade will be automated because the latest version will be used when the service is called at runtime.

WS implementation also requires a loosely coupled architecture where new services can be added at runtime and old services can be replaced. Furthermore, vendors will compete to supply most dependable and/or marketable services on the web, and this also changes the way software industries earn their revenue.

Quality assurance as well as security and privacy will be important for both service clients and providers including those serve as intermediate agents such as UDDI servers.

WS provide a new way for globalization where companies, regardless of their background such as nationalities, languages, and culture, must now compete in a global market where the only rule is interoperability via architecture and interfaces. It is no longer possible to have local or national markets where local companies can do well due to market segmentation. If a company does not compete in the service market globally, it will wither as soon as new replacement services are published on the web. This will increase the pace of global competition, and the companies that have the great software technology will win instead of the one that has the great financial resources only.

Concepts of WS is far beyond software, in the future, hardware will also have a corresponding service where hardware vendors will supply new components to fit into existing well-published architecture for specific applications.

## Invited Talk

# An Exact Reliability Block Diagram Calculation Tool to Design Very Complex Systems

Mehmet Şahinoglu

Fellow SDPS, Senior Member IEEE, ASA, Elected ISI Member  
Eminent Scholar & Chairman-Professor of Department of Computer & Information Science  
Troy State University Montgomery, Alabama  
[mesa@tsum.edu](mailto:mesa@tsum.edu)  
<http://www.tsum.edu/~mesa>

In this study, the novel ERBDC method is introduced as a tool both to compute exact (s-t) reliability for a network, which is important in the design of even moderately sized networks, and to redesign an existing network. This talk examines very complex network designs or imbedded systems such as in the topology of an Internet service grid or a chip design for calculating s-node (source) to t-node (target) availability. The ERBDC method is a tool which reduces any complex system to a series reliability block diagram, by first finding all existing paths and then trimming all the redundant or protruding component duplications through an algorithm to finally calculate an exact reliability, favorably compared to conventionally employed upper bound calculations (theoretically feasible but practically infeasible) with respect to path set and cut set formulations. All node to node availabilities are calculated. Then, weaker node to node paths are then reinforced and the resulting network designs are compared to the original ones in terms of node to node availability.

All current exact computational algorithms for general networks are based on enumeration of states, minpaths or mincuts. However, the exponential size of enumeration of tie sets or cut sets quickly becomes unmanageable as size (links and nodes) of the network grows. Hence, network reliability bounds, estimation or simulation is commonly used for non-trivial sized networks. The method used in this paper, Exact Reliability Block Diagram Calculation is an exact calculation tool for s-t reliability but tractable for large networks. As an example of this method, a Java applet first examines a simple LAN of 6 nodes and 7 links in the sample problem 1. Then, a complex network exemplifying a popular WAN operation will be shown consisting of 32 connections and 19 nodes as in the sample problem 2. Also, an application of the Sahinoglu-Libby probability distribution function (otherwise known as G3B) will be illustrated through a Java applet on how to sample historical failure and repair data to estimate the node and link reliabilities for the implementation of the ERBDC algorithm.

The proposed method allows designers to see and investigate all possible paths between all s-t nodes in a network as well as computing exact s-t reliabilities to make sensitivity analysis on any design obtained by a heuristic design algorithm. Thus, designers can reinforce their designs considering both reliabilities and costs of all components and links in an existing network being redesigned. The newly proposed

## 6 An Exact Reliability Block Diagram Calculation Tool to Design Very Complex Systems

ERBDC method opens a new avenue to overcome the impasse of calculating the exact s-t reliabilities of very large complex systems, hence paving the way to conveniently and accurately redesigning and improving computer network reliability. This will consequently improve the quality of distributed network systems while mitigating the effects of poor software security.

# Computer-Aided Refinement of Data Structures on Higher-Order Algebraic Specifications

Walter Dosch<sup>1</sup> and Sönke Magnussen<sup>2</sup>

<sup>1</sup> Institute of Software Technology and Programming Languages  
University of Lübeck

Ratzeburger Allee 160, D-23538 Lübeck, Germany

<sup>2</sup> Lufthansa Revenue Services, Application Management  
Schuetzenwall 1, D-22844 Norderstedt, Germany

**Abstract.** The paper studies the transformational refinement of data structures in the framework of higher-order algebraic specifications. We present novel procedures that mechanize the refinement of entire data structures within a single complex transformation step. The transformations validate a general refinement relation that captures different types of simulations. General transformation rules describe algebraic implementations based on abstraction and representation functions. Specialized transformations cover particular changes between data structures. All transformation procedures have been implemented in the Lübeck Transformation System. The system uses analysis algorithms to establish the soundness conditions of the transformations by syntactic criteria. We report practical experiences about manipulating data structures with the system. The paper summarizes results from the second author's PhD thesis [20].

**Keywords:** Higher-order algebraic specification, refinement of data structure, algebraic implementation, transformation system

## 1 Introduction

Formal methods offer a secure development process for software and hardware systems. The transformational approach [15] organizes the development as a sequence of rule-based steps refining a behavioural specification to an efficient realization. The derived implementation is correct by construction without a posteriori verification if the development uses sound transformation rules [2] respecting a semantic refinement relation.

Transformation systems assist the programmer with various degrees of interaction. As standard services, they offer the safe manipulation of specifications following elementary transformation rules. Advanced transformation systems also support larger transformation steps where the user can concentrate on the essential design decisions. Chaining wide-spanned transformations results in compact derivations with an adequate degree of mechanization.

The mechanization of the refinement of data structures in a transformation system faces both theoretical and practical problems. Combined data structures built by several basic modules along with functions operating on these combined data structures form

complex entities which must carefully be transformed by syntactic manipulations. The refinement of data structures cannot be confined to transformations for single constituents; rather refinement steps generally affect the entire data structure and dependent functions. Further problems arise from complex semantic conditions that constrain many refinement steps to ensure their soundness. Here advanced analysis algorithms are needed to implement powerful sufficient syntactic criteria. The transformation of data structures has to be well understood, since it lays a semantic basis for transforming classes in object-oriented software design.

In this paper we study the refinement of data structures for higher-order algebraic specifications with the particular aim to mechanize the refinement relation by transformations. The approach is based on a novel refinement relation between specifications. It requires that all models of the refined specification can be retrieved as subalgebras of quotients of models from the original specification. This notion of refinement generalizes the model inclusion from predicate logic [12] to capture flexible changes between data structures.

The refinement process can be mechanized to a certain degree; it has been implemented in the Lübeck Transformation System — a tool for the interactive development of software [9,20]. We present several mechanizable transformations for data structures satisfying the semantic refinement relation. The application conditions are either verified automatically by analysing the specification or inserted as axioms into the specification. These axioms serve as proof obligations which have to be handled during the development. The transformations capture the general approach of algebraic implementations which can be specialized for particular situations like constructor enrichments or constructor implementations.

Section 2 presents motivating examples which illustrate characteristic refinement steps for data structures. In Section 3 we survey the foundations of our approach to higher-order algebraic specifications. We concentrate on constructor generated models supporting a mechanization of the refinement steps. The general method of algebraic implementation is also applicable for non constructor generated specifications. Section 4 first presents a general framework for the refinement of data structures using representation and abstraction functions. Then we establish that the four different types of simulations validate the refinement relation. Section 5 first presents a general transformation rule mechanizing the algebraic implementation of data structures, and then studies various special cases. We illustrate these transformation rules by revisiting the introductory examples. Section 6 surveys the refinement of data structures from a user’s view point as it is handled in the Lübeck Transformation System.

Throughout the paper we assume that the reader is familiar with the basic notions of algebraic specifications [15,23] and data refinement [7].

## 2 Introductory Examples

Algebraic specifications define data structures in an abstract way by naming the required sorts and function symbols together with their properties. During software development, the abstract sorts must be represented by the concrete types of a programming language; the function symbols must correctly be realized by algorithms in terms of the basic functions provided.

The refinement of algebraic specifications should document the design decisions made by successively adding implementation details to the abstract specification. There are general methods and specialized techniques for refining an abstract specification towards a concrete implementation [5].

In this introductory section, we illustrate important refinement steps for algebraic specifications presenting simple, yet characteristic cases. The examples exhibit the changes which a data structure undergoes during various refinement steps. The illustrating examples will be revisited in the formal treatment to follow.

## 2.1 Basic Data Structures

We present algebraic specifications for basic data structures like truth values and natural numbers to familiarize the reader with the notation. These specifications will be used as basic modules for more building complex specifications [6].

**Truth Values.** The specification *Bool* for the truth values defines the constants *true* and *false* along with the usual Boolean operators.

```

spec   Bool =
sorts  bool = true | false
ops    not : (bool)bool
       and : (bool, bool)bool
       or  : (bool, bool)bool
vars   a, b : bool
axioms
       not1 : not(true) = false
       not2 : not(false) = true
       and1 : and(true, b) = b
       and2 : and(false, b) = false
       or1  : or(a, b) = not(and(not(a), not(b)))
end

```

**Natural Numbers.** The specification *Nat* defines natural numbers using the constructors *zero* and *succ*. As a short-hand notation, we define the sort *nat* by a recursive sort declaration. The specification *Nat* extends the specification *Bool* which is needed for the equality predicate.

```

spec Nat = Bool ;
sorts  nat = zero | succ(nat)
ops   eqNat : (nat, nat)bool
      add  : (nat, nat)nat
      mult : (nat, nat)nat

```

```

vars       $m, n : \text{nat}$ 
axioms
   $\text{eqNat1} : \text{eqNat}(\text{zero}, \text{zero}) = \text{true}$ 
   $\text{eqNat2} : \text{eqNat}(\text{zero}, \text{succ}(m)) = \text{false}$ 
   $\text{eqNat3} : \text{eqNat}(\text{succ}(m), \text{zero}) = \text{false}$ 
   $\text{eqNat4} : \text{eqNat}(\text{succ}(m), \text{succ}(n)) = \text{eqNat}(m, n)$ 
   $\text{add1} : \text{add}(m, \text{zero}) = m$ 
   $\text{add2} : \text{add}(m, \text{succ}(n)) = \text{succ}(\text{add}(m, n))$ 
   $\text{mult1} : \text{mult}(m, \text{zero}) = \text{zero}$ 
   $\text{mult2} : \text{mult}(m, \text{succ}(n)) = \text{add}(m, \text{mult}(m, n))$ 
end

```

The specification *Nat* is algorithmic, since all operations are defined by structural induction over the sort *nat*.

## 2.2 Implementing Sets by Boolean Arrays

In this subsection, we refine the data structure ‘sets of natural numbers’ to a realization using Boolean arrays. The transformation illustrates three characteristic refinement steps, viz. the introduction of a generation constraint, the reduction of a constructor system, and the implementation of a constructor sort by another constructor sort.

**Sets of Natural Numbers.** The specification *FinSet* describes sets of natural numbers. The operation *addElem* inserts an element into a set, *union* forms the set union, and *memb* tests whether an element is contained in a set. The sort *set* is specified as a loose sort imposing no generation constraint.

```

spec    $\text{FinSet} = \text{Bool} + \text{Nat} ;$ 
sorts     $\text{set}$ 
ops      $\text{emptySet} : \text{set}$ 
         $\text{addElem} : (\text{nat}, \text{set})\text{set}$ 
         $\text{union} : (\text{set}, \text{set})\text{set}$ 
         $\text{memb} : (\text{nat}, \text{set})\text{bool}$ 
vars     $m, n : \text{nat}, r, s, t : \text{set}$ 
axioms
   $\text{union}_{\text{comm}} : \text{union}(r, s) = \text{union}(s, r)$ 
   $\text{union}_{\text{assoc}} : \text{union}(\text{union}(r, s), t) = \text{union}(r, \text{union}(s, t))$ 
   $\text{union1} : \text{union}(s, \text{emptySet}) = s$ 
   $\text{union2} : \text{union}(s, \text{addElem}(m, t)) = \text{addElem}(m, \text{union}(s, t))$ 
   $\text{memb1} : \text{memb}(m, \text{emptySet}) = \text{false}$ 
   $\text{memb2} : \text{memb}(m, \text{addElem}(n, s)) = \text{or}(\text{eqNat}(m, n), \text{memb}(m, s))$ 
end

```

**Introducing a Generation Constraint.** Heading towards an algorithmic axiomatization, we first impose a generation constraint on the sort *set* confining the corresponding

carrier to *finite sets* of natural numbers. In a straightforward approach, we can take the collection of all function symbols with result sort *set* as the constructor system. We refine the loose sort *set* in specification *FinSet* to a constructor sort introducing a recursive sort declaration:

```
spec FinSet = Bool + Nat ;
sorts set = emptySet | addElem(nat, set) | union(set, set)
ops memb : (nat, set)bool
vars
:
end
```

Each interpretation of the carrier of the sort *set* requires to be finitely generated by the three constructors *emptySet*, *addElem*, and *union*.

**Reducing a Constructor System.** A closer look at the axioms of specification *FinSet* reveals that each formation of a set involving the constructor *union* can be transformed to a formation using solely the constructors *emptySet* and *addElem*. Thus we can reduce the constructor system dropping the constructor *union* in the next refinement step:

```
spec FinSet = Bool + Nat ;
sorts set = emptySet | addElem(nat, set)
ops union : (set, set)set
      memb : (nat, set)bool
vars
:
end
```

**Constructor Translation.** After these two refinement steps, the resulting specification *FinSet* is algorithmic, yet the constructors prevent an efficient implementation of the functions involved. Therefore we aim at implementing the data structure ‘finite sets of natural numbers’ by Boolean arrays.

To this end, we introduce the specification *BoolArray* defining arrays of truth values indexed by natural numbers. The operation *put* writes a Boolean value to an array at a given position, the operation *lookup* reads the element stored at a position.

```
spec BoolArray = Bool + Nat ;
sorts array = init | put(array, nat, bool)
ops lookup : (nat, array)bool
vars b, c : bool, m, n : nat, a : array
axioms
  put1 : put(put(a, m, b), n, c) =
    if eqNat(m, n) then put(a, n, c) else put(put(a, n, c), m, b)

  lookup1 : lookup(m, init) = false
  lookup2 : lookup(m, put(a, n, b)) = if eqNat(m, n) then b else lookup(m, a)
end
```

Now we embed the abstract sort *set* into the concrete sort *array* by translating all constructor terms of sort *set* into terms of sort *array*. To this end we specify the sort *set* as *array* and describe the translation by the equations

$$\begin{aligned} \text{emptySet} &= \text{init} \\ \text{addElem}(m, s) &= \text{put}(s, m, \text{true}) . \end{aligned}$$

As a consequence the former constructors *emptySet* and *addElem* become operations. Then we replace all occurrences of the former constructor terms of sort *set* in the axioms by a translated term of sort *array* and keep the remaining term structures. With this refinement step, we obtain the following result:

```
spec   FinSet = Bool + Nat + BoolArray ;
sorts    set = array
ops     emptySet : set
        addElem  : (nat, set)set
        union    : (set, set)set
        memb    : (nat, set)bool
vars   m, n : nat, r, s, t : set
axioms
    emptySet1 : emptySet = init
    addElem1  : addElem(m, s) = put(s, m, true)
    unioncomm : union(r, s) = union(s, r)
    unionassoc : union(union(r, s), t) = union(r, union(s, t))
    union1   : union(s, init) = s
    union2   : union(s, put(t, m, true)) = put(union(s, t), m, true)
    memb1   : memb(m, init) = false
    memb2   : memb(m, put(s, n, true)) = or(eqNat(m, n), memb(m, s))
end
```

This example illustrates that the implementation of data structures involves refinement steps touching different algebraic notions. This small scale example also shows the complexity of the refinement steps which manifests the need both for a unifying theory and a reliable interactive tool. The reader is invited to trace the sequence of elementary syntactic manipulations that are needed to completely mechanize the refinement steps sketched so far.

### 2.3 Implementing Stacks by Arrays with Pointers

The preceding subsection illustrated three special refinement steps for data structures which leave the signature of the specification unaffected. In the second example we now address the general notion of algebraic implementation. We revisit the well-known implementation of stacks by pairs of an array and a pointer. In this refinement step, a constructor sort is embedded into a derived sort built from two constructor sorts. In the sequel we concentrate on the manipulations necessary to perform this change of the data structure in a sound way.

**Stacks of Natural Numbers.** The specification *Stack* defines stacks with elements of sort *elem*. Stacks are constructed from the empty stack *emptyStack* by successively *pushing* elements to the stack. The operation *top* reads, the operation *pop* removes the first element of a nonempty stack.

```

spec   Stack = Nat ;
sorts   elem
        stack = emptyStack | push(elem, stack)
ops    error : elem
        top  : (stack)elem
        pop  : (stack)stack
vars   e : elem, s : stack
axioms
        top1 : top(emptyStack) = error
        top2 : top(push(e, s)) = e
        pop1 : pop(emptyStack) = emptyStack
        pop2 : pop(push(e, s)) = s
end

```

**Arrays.** The implementation of stacks by arrays is based on the following specification *ElemArray* defining arrays of elements indexed with natural numbers:

```

spec   ElemArray = Bool + Nat ;
sorts   elem
        array = init | put(array, nat, elem)
ops    error : elem
        lookup : (nat, array)elem
vars   e, f : elem, m, n : nat, a : array
axioms
        put1 : put(put(a, m, e), n, f) = if eqNat(m, n) then put(a, n, f)
                                                else put(put(a, n, f), m, e)
        lookup1 : lookup(m, init) = error
        lookup2 : lookup(m, put(a, n, e)) = if eqNat(m, n) then e else lookup(m, a)
end

```

**Constructing an Algebraic Implementation.** For implementing stacks by pairs of an array and a natural number, we first introduce a new sort name for the chosen representation:

```
sorts   stacknew = (nat, array)
```

The relationship between the abstract sort *stack* and the concrete sort *stack<sub>new</sub>* is defined by a representation function and/or an abstraction function [17]. These functions relate

abstract stacks constructed by *emptyStack* and *push* to their representation as pairs (*nat*, *array*):

```
ops    repr : (stack)stacknew
      abstr : (stacknew)stack
```

For each function symbol involving the sort *stack*, we introduce a corresponding function symbol operating on the implementation sort *stack<sub>new</sub>*:

```
ops    emptyStacknew : stacknew
      pushnew : (elem, stacknew)stacknew
      topnew : (stacknew)elem
      popnew : (stacknew)stacknew
```

The functions on the implementation level must show a similar behaviour as the functions on the abstract level. The relation between the abstract and the concrete function can be established using, for example, the abstraction function with the following axioms:

```
vars   e : elem, s : stacknew
axioms
      emptyStacknew : abstr(emptyStacknew) = emptyStack
      pushnew : abstr(pushnew(e, s)) = push(e, abstr(s))
      topnew : topnew(s) = top(abstr(s))
      popnew : abstr(popnew(s)) = pop(abstr(s))
```

Using standard derivation steps from transformational programming like fold, unfold and case analysis, these equations can be transformed into an algorithmic form, if a suitable axiomatization of the abstraction function is provided. Finally, the derivation reaches the following specification after renaming the constituents:

```
spec StackByArray = Bool + Nat + ElemArray ;
sorts
      elem
      stack = (nat, array)
ops
      error : elem
      emptyStack : stack
      push : (elem, stack)stack
      top : (stack)elem
      pop : (stack)stack
vars e : elem, m : nat, a : array, s : stack
axioms
      emptyStack1 : emptyStack = (zero, init)
      push1 : push(e, (m, a)) = (succ(m), put(a, succ(m), e))
      top1 : top(zero, a) = error
      top2 : top(succ(m), a) = lookup(succ(m), a)
      pop1 : pop(zero, a) = (zero, init)
      pop2 : pop(succ(m), a) = (m, a)
end
```

The data structure refinement presented in this subsection constitutes an algebraic implementation to be discussed more formally in Section 4.

We will extend the common approach to data structure refinement from first-order to higher-order functions. Moreover, we provide powerful transformation rules that refine data structures by manipulating algebraic specification in a sound way. For the accompanying conditions, we implemented analysis algorithms which establish the desired properties for a wide class of specifications.

## 3 Foundations

We survey the basic concepts of our approach to higher-order algebraic specifications to render the subsequent results precise.

As specification language we use algebraic specifications with a loose constructor generated semantics [24]. The theory is extended to higher-order sorts and higher-order terms for expressing advanced concepts of functional programming. The approach, similar to [21] and [16], imposes a generation constraint on sorts to support the constructor based specification of data types. Furthermore we introduce subalgebras [22] and congruences on the carriers of algebras [3] aiming at a refinement relation suitable for data structure refinements. Both concepts, subalgebras and congruences are extended to the higher-order case.

### 3.1 Signatures

Signatures characterize the interface of a specification by naming its constituents.

For a nonempty set  $S$  of *basic sorts*, the set  $S^{\times, \rightarrow}$  of *derived sorts* comprises  $S$ , all *tuple sorts*  $(s_1, \dots, s_n)$  for  $n \geq 2$ , and all *function sorts*  $(s_1 \rightarrow s_2)$  defined inductively over  $S$ .

A *signature*  $\Sigma = (S, F, C)$  consists of

- a set  $S = SC \dot{\cup} SP$  of basic sorts composed from a set  $SC$  of *constructor sorts* and a set  $SP$  of *parameter sorts*,
- an  $S^{\times, \rightarrow}$ -indexed family  $F$  of mutually disjoint sets  $F_s$  of *function symbols* with functionality  $s \in S^{\times, \rightarrow}$ ,
- an  $S^{\times, \rightarrow}$ -indexed subfamily  $C \subseteq F$  of *constructors*  $C_s$  with either  $s \in SC$  or  $s = (s_1 \rightarrow sc)$  and  $sc \in SC$ .

For a function symbol  $f \in F_{(s_1 \rightarrow s_2)}$  we call  $s_1$  the *argument sort* and  $s_2$  the *result sort* of  $f$ . The function symbols in  $F \setminus C$  are called *operations*. In the following we will denote basic sorts by  $sb$ , constructor sorts by  $sc$ , and derived sorts by  $s$ . We write  $\Sigma = (S, F)$  for signatures whenever the family of constructors is not relevant in the respective context.

A signature  $\Sigma_1 = (S_1, F_1)$  is called a *subsignature* of a signature  $\Sigma_2 = (S_2, F_2)$ , denoted by  $\Sigma_1 \subseteq \Sigma_2$ , if  $S_1 \subseteq S_2$  and  $F_1 \subseteq F_2$  holds.

### 3.2 Signature Morphisms

Signature morphisms rename sort and function symbols in a compatible way.

Let  $S_1, S_2$  be two sets of basic sorts. A *sort mapping*  $\iota : S_1 \rightarrow S_2^{\times, \rightarrow}$  is extended to a *sort morphism*  $\iota : S_1^{\times, \rightarrow} \rightarrow S_2^{\times, \rightarrow}$  by setting

$$\begin{aligned}\iota((s_1, \dots, s_n)) &= (\iota(s_1), \dots, \iota(s_n)) \\ \iota(s_1 \rightarrow s_2) &= (\iota(s_1) \rightarrow \iota(s_2)).\end{aligned}$$

A sort morphism can be propagated to signatures by joining it with a compatible mapping for the function symbols. Let  $\Sigma_1 = (S_1, F_1)$  and  $\Sigma_2 = (S_2, F_2)$  be signatures. A *signature morphism*  $\delta : \Sigma_1 \rightarrow \Sigma_2$  consists of a sort morphism  $\delta : S_1^{\times, \rightarrow} \rightarrow S_2^{\times, \rightarrow}$  and an (equally denoted)  $S_1^{\times, \rightarrow}$ -indexed family  $\delta : F_1 \rightarrow F_2$  of functions  $\delta_s : (F_1)_s \rightarrow (F_2)_{\delta(s)}$ . In general, the image of a signature under a signature morphism does not form a signature. Therefore we define the *image signature*  $\langle \delta(\Sigma_1) \rangle$  as the smallest signature comprising  $\delta(\Sigma_1)$ .

### 3.3 Algebras

Algebras provide an interpretation for signatures.

A  $\Sigma$ -algebra  $\mathcal{A} = ((sb^{\mathcal{A}})_{sb \in S}, (f^{\mathcal{A}})_{f \in F})$  consists of a nonempty *carrier set*  $sb^{\mathcal{A}}$  for every basic sort  $sb \in S$  and an element  $f^{\mathcal{A}} \in s^{\mathcal{A}}$  for every function symbol  $f \in F_s$ .

The interpretation of derived sorts in  $S^{\times, \rightarrow}$  is defined inductively: a tuple sort  $(s_1, \dots, s_n)^{\mathcal{A}} = s_1^{\mathcal{A}} \times \dots \times s_n^{\mathcal{A}}$  ( $n \geq 2$ ) denotes the Cartesian product and a function sort  $(s_1 \rightarrow s_2)^{\mathcal{A}} = [s_1^{\mathcal{A}} \rightarrow s_2^{\mathcal{A}}]$  the set of all total functions.

A  $\Sigma$ -algebra  $\mathcal{A}$  with signature  $\Sigma = (S, F, C)$  is called *constructor generated*, if for all constructor sorts  $sc \in SC$  the carrier set  $sc^{\mathcal{A}}$  is the least set wrt. set inclusion satisfying

$$C_{sc}^{\mathcal{A}} \subseteq sc^{\mathcal{A}} \text{ and } c(s_1^{\mathcal{A}}) \subseteq sc^{\mathcal{A}} \text{ for all } c \in C_{(s_1 \rightarrow sc)}.$$

The  $\Sigma_1$ -reduct  $\mathcal{A}|_{\Sigma_1} = ((s^{\mathcal{A}|_{\Sigma_1}})_{s \in S_1}, (f^{\mathcal{A}|_{\Sigma_1}})_{f \in F_1})$  of a given  $\Sigma_2$ -algebra  $\mathcal{A} = ((s^{\mathcal{A}})_{s \in S_2}, (f^{\mathcal{A}})_{f \in F_2})$  wrt. a subsignature  $\Sigma_1 \subseteq \Sigma_2$  keeps all carriers and functions of the subsignature, that is  $s^{\mathcal{A}|_{\Sigma_1}} = s^{\mathcal{A}}$  for all  $s \in S_1$  and  $f^{\mathcal{A}|_{\Sigma_1}} = f^{\mathcal{A}}$  for all  $f \in F_1$ .

### 3.4 Terms and Equations

Terms are well-formed expressions built from the function symbols of a signature and variables. Equations specify properties by relating terms of equal sort.

An  $S^{\times, \rightarrow}$ -indexed family  $X$  of *variables* consists of mutually disjoint sets  $X_s$  of variables of sort  $s$ .

Given a signature  $\Sigma = (S, F)$  and an  $S^{\times, \rightarrow}$ -indexed family  $X$  of variables, the  $S^{\times, \rightarrow}$ -indexed family  $T(\Sigma, X)$  of  $\Sigma$ -terms consists of the inductively defined sets  $T_s(\Sigma, X)$  of  $\Sigma$ -terms of sort  $s$ :

- (1)  $X_s \subseteq T_s(\Sigma, X)$  (variables)
- (2)  $F_s \subseteq T_s(\Sigma, X)$  (function symbols)
- (3) If  $n \geq 2$  and  $t_i \in T_{s_i}(\Sigma, X)$  for all  $1 \leq i \leq n$ , then  $(t_1, \dots, t_n) \in T_{(s_1, \dots, s_n)}(\Sigma, X)$ . (tuple)
- (4) If  $t_f \in T_{(s_1 \rightarrow s_2)}(\Sigma, X)$  and  $t_a \in T_{s_1}(\Sigma, X)$ , then  $(t_f(t_a)) \in T_{s_2}(\Sigma, X)$ . (function application)
- (5) If  $t_f \in T_{(s_1 \rightarrow s_2)}(\Sigma, X)$  and  $t_g \in T_{(s_2 \rightarrow s_3)}(\Sigma, X)$ , then  $(t_g \circ t_f) \in T_{(s_1 \rightarrow s_3)}(\Sigma, X)$ . (function composition)

For  $X = \emptyset$  we obtain the family  $T(\Sigma)$  of *ground terms*.

In writing terms, we drop brackets with usual conventions. Moreover, we unify the treatment of constants  $f \in F_s$  and unary function symbols by identifying  $f()$  with  $f$ .

A  $\Sigma$ -equation is a universally quantified formula  $\forall x_1: s_1 \dots \forall x_n: s_n$  ( $l = r$ ) relating two terms  $l, r \in T_s(\Sigma, X)$  of the same sort  $s \in S^{\times, \rightarrow}$ . When writing formulae, the quantification ranging over all variables in  $l$  and  $r$  can be omitted.

### 3.5 Algebraic Specifications and Models

Specifications define algebras in an axiomatic way [13] using the characteristic properties of their functions.

An (*algebraic*) specification  $(\Sigma, E)$  is composed of a signature  $\Sigma$  and a set  $E$  of  $\Sigma$ -equations. A  $\Sigma$ -algebra satisfying all equations in  $E$  is called a *model* of the specification  $(\Sigma, E)$ . The class of models of  $(\Sigma, E)$  is denoted by  $MOD(\Sigma, E)$ . The *semantics*  $CGEN(\Sigma, E)$  of an algebraic specification  $(\Sigma, E)$  comprises the class of all constructor generated models.

### 3.6 Subalgebras and Congruences

A homomorphism is a family of mappings between the carrier sets of two  $\Sigma$ -algebras which is compatible with all operations [14].

Let  $\mathcal{A}$  and  $\mathcal{B}$  be  $\Sigma$ -algebras with the same signature  $\Sigma = (S, F)$ . A  $\Sigma$ -homomorphism  $\varphi : \mathcal{A} \rightarrow \mathcal{B}$  is an  $S^{\times, \rightarrow}$ -indexed family of mappings  $\varphi_s : s^{\mathcal{A}} \rightarrow s^{\mathcal{B}}$  with

- $\varphi_s(f^{\mathcal{A}}) = f^{\mathcal{B}}$  for all  $f \in F_s$ ,
- $\varphi_{(s_1, \dots, s_n)}((a_1, \dots, a_n)) = (\varphi_{s_1}(a_1), \dots, \varphi_{s_n}(a_n))$  for all  $a_i \in s_i^{\mathcal{A}}, 1 \leq i \leq n$ ,
- $\varphi_{s_2}(f(a)) = \varphi_{(s_1 \rightarrow s_2)}(f)(\varphi_{s_1}(a))$  for all  $f \in (s_1 \rightarrow s_2)^{\mathcal{A}}$  and  $a \in s_1^{\mathcal{A}}$ .

An algebra can be embedded into a “larger” algebra that comprises supersets for all its carriers. A  $\Sigma_1$ -algebra  $\mathcal{A}$  with  $\Sigma_1 = (S_1, F_1)$  is said to be *embedded* into a  $\Sigma_2$ -algebra  $\mathcal{B}$  via a sort morphism  $\iota : S_1^{\times, \rightarrow} \rightarrow S_2^{\times, \rightarrow}$  (written  $\mathcal{A} \overset{\iota}{\subseteq} \mathcal{B}$ ), if  $sb^{\mathcal{A}} \subseteq \iota(sb)^{\mathcal{B}}$  holds for all basic sorts  $sb \in S_1$ . The inclusion relation for basic sorts induces further properties of the derived sorts. The inclusion relation can not be transferred to function types, since the equality of functions demands equal domains and ranges. Therefore we introduce a particular equivalence notion.

Let the  $\Sigma_1$ -algebra  $\mathcal{A}$  be embedded in the  $\Sigma_2$ -algebra  $\mathcal{B}$  via the sort morphism  $\iota : S_1^{\times, \rightarrow} \rightarrow S_2^{\times, \rightarrow}$ . The *equivalence on embedded sorts*  ${}^{\mathcal{A}} \asymp_s {}^{\mathcal{B}}$  is a family of  $S^{\times, \rightarrow}$ -indexed relations  ${}^{\mathcal{A}} \asymp_s {}^{\mathcal{B}} \subseteq s^{\mathcal{A}} \times \iota(s)^{\mathcal{B}}$  with

- $a {}^{\mathcal{A}} \asymp_{sb} {}^{\mathcal{B}} b \Leftrightarrow a = b$
- $(a_1, \dots, a_n) {}^{\mathcal{A}} \asymp_{(s_1, \dots, s_n)} {}^{\mathcal{B}} (b_1, \dots, b_n) \Leftrightarrow a_i {}^{\mathcal{A}} \asymp_{s_i} {}^{\mathcal{B}} b_i$  for all  $1 \leq i \leq n$
- $f {}^{\mathcal{A}} \asymp_{(s_1 \rightarrow s_2)} {}^{\mathcal{B}} g \Leftrightarrow \forall a_1 \in s_1^{\mathcal{A}} \exists a_2 \in \iota(s_1)^{\mathcal{B}} a_1 {}^{\mathcal{A}} \asymp_{s_2} {}^{\mathcal{B}} a_2 \wedge f(a_1) {}^{\mathcal{A}} \asymp_{s_2} {}^{\mathcal{B}} g(a_2)$ .

Based on the definition of embedded algebras we introduce subalgebras. Given two signatures  $\Sigma_1 = (S_1, F_1)$  and  $\Sigma_2 = (S_2, F_2)$ , a  $\Sigma_1$ -algebra  $\mathcal{A}$  is said to be a *subalgebra* of a  $\Sigma_2$ -algebra  $\mathcal{B}$  via a signature morphism  $\delta : \Sigma_1 \rightarrow \Sigma_2$  (written  $\mathcal{A} \overset{\delta}{\subseteq} \mathcal{B}$ ) if we have

$\mathcal{A} \stackrel{\delta}{\sqsubseteq} \mathcal{B}$  and  $f^{\mathcal{A}} \approx_s^{\mathcal{B}} \delta(f)^{\mathcal{B}}$  for all  $f \in F_{1,s}$ . In the following we will use the relation  $\stackrel{\delta}{\sqsubseteq}$  modulo isomorphic algebras.

A congruence is an equivalence relation on the carrier sets of an algebra which is compatible with tupling and function application. A  $\Sigma$ -congruence  $\approx^{\mathcal{A}}$  on a  $\Sigma$ -algebra  $\mathcal{A}$  with  $\Sigma = (S, F)$  is an  $S^{\times, \rightarrow}$ -indexed family  $\approx^{\mathcal{A}}$  of equivalence relations  $\approx_s^{\mathcal{A}}$  on  $s^{\mathcal{A}}$  validating

- If  $a_i \approx_{s_i}^{\mathcal{A}} b_i$  for  $1 \leq i \leq n$ , then  $(a_1, \dots, a_n) \approx_{(s_1, \dots, s_n)}^{\mathcal{A}} (b_1, \dots, b_n)$ .
- If  $g \approx_{(s_1 \rightarrow s_2)}^{\mathcal{A}} h$  and  $a \approx_{s_1}^{\mathcal{A}} b$ , then  $g(a) \approx_{s_2}^{\mathcal{A}} h(b)$ .

A congruence can be used to build the quotient algebra using the equivalence classes as carrier sets. Let  $\mathcal{A}$  be a  $\Sigma$ -algebra with  $\Sigma = (S, F)$  and  $\approx^{\mathcal{A}}$  a  $\Sigma$ -congruence on  $\mathcal{A}$ . The quotient algebra  $\mathcal{A}/\approx = (S^{\mathcal{A}/\approx}, F^{\mathcal{A}/\approx})$  is given by

$$\begin{aligned} s^{\mathcal{A}/\approx} &= s^{\mathcal{A}} / \approx_s^{\mathcal{A}} \\ f^{\mathcal{A}/\approx}([a]_{\approx_{s_1}^{\mathcal{A}}}) &= [f^{\mathcal{A}}(a)]_{\approx_{s_2}^{\mathcal{A}}} . \end{aligned}$$

Since  $\approx$  is a congruence, the functions of the quotient algebra are well defined, and the structure  $\mathcal{A}/\approx$  forms a  $\Sigma$ -algebra. A congruence can be used to reduce an algebra to the behaviour viewed modulo a congruence [4].

A  $\Sigma_1$ -algebra  $\mathcal{A}$  is *congruent to a  $\Sigma_2$ -algebra  $\mathcal{B}$  via a signature morphism  $\delta : \Sigma_1 \rightarrow \Sigma_2$*  (written  $\mathcal{A} \stackrel{\delta}{\approx} \mathcal{B}$ ) if there is a  $\Sigma_2$ -congruence  $\approx^{\mathcal{B}}$  such that  $sb^{\mathcal{A}} = \delta(sb)^{\mathcal{B}} / \approx_{sb}^{\mathcal{B}}$  for all  $sb \in S_1$  and  $f^{\mathcal{A}} = [\delta(f)^{\mathcal{B}}]_{\approx_s^{\mathcal{B}}}$  for all  $s \in S_1^{\times, \rightarrow}$ , and  $f \in (F_1)_s$ . Again we use this relation modulo isomorphic algebras.

### 3.7 Refinement

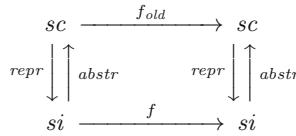
The refinement of a specification enriches its signature while retaining the properties in its models.

A specification  $(\Sigma_1, E_1)$  is *refined to a specification  $(\Sigma_2, E_2)$* , if for all  $\mathcal{B} \in CGEN(\Sigma_2, E_2)$  there is  $\mathcal{A} \in CGEN(\Sigma_1, E_1)$  and a  $\Sigma_2$ -congruence  $\approx^{\mathcal{B}}$  such that  $\mathcal{A} \stackrel{\delta}{\sqsubseteq} \mathcal{B}/\approx$ . We abbreviate the refinement relation by  $(\Sigma_1, E_1) \xrightarrow{\delta} (\Sigma_2, E_2)$ .

So the properties of the original specifications can be retrieved from the models of the refined specification. Simple refinement steps are for example enrichments of the signature, the addition of axioms, term rewriting and induction. Compared to other approaches [19,18], our implementation relation has no forget operator and imposes a generation constraint.

## 4 The Refinement of Data Structures

In this section we survey a general model theoretic framework for the refinement of data structures. Based on the idea of representation and abstraction functions, we present new mechanizable transformations for data structures defined by higher-order algebraic specifications.



**Fig. 1.** Data structure refinement for a unary homogenous function

## 4.1 A Framework for Data Structure Refinement

A data structure refinement embeds the carrier set of an *abstract sort* into the carrier set of an *implementation sort*. The abstract sort is a basic sort of the original specification, the implementation sort is a derived sort in the refined specification.

The data structure refinement is settled by a signature morphism mapping abstract sorts to their implementation sorts. Moreover, the abstract function symbols operating on abstract sorts are mapped to the implementation function symbols operating on the implementation sorts. In this way, a data structure refinement transforms an entire specification according to a signature morphism.

The refinement of data structures must satisfy the refinement relation between specifications. Therefore the described transformation must not change the behaviour of the implementation functions operating on the implementation sorts. This is guaranteed by refining the implementation function symbols in terms of the abstract function symbols using two auxiliary functions viz. representation and abstraction functions. Hence we simulate the abstract function symbols by the implementation function symbols and use the auxiliary functions for converting the arguments between abstract and implementation sorts. This principle was first introduced by Hoare [17] and later used for (first order) algebraic specifications by Broy [5].

Our approach is driven by the aim of mechanizing such refinements, i.e. we head for transformation rules on entire specifications that embody data structure refinements. Moreover, we extend the approach to higher-order sorts and function symbols.

Fig. 1 illustrates the basic idea of representation and abstraction functions. The abstract sort  $sc$  is refined to the implementation sort  $si$ , and the function symbol  $fold$  is refined to the function symbol  $f$ . The refinement is described by a representation function  $repr : sc \rightarrow si$  mapping each element of the abstract sort to a corresponding element of the implementation sort, and/or by an abstraction function  $abstr : si \rightarrow sc$  mapping vice versa elements of the implementation sort to elements of the abstract sort. These auxiliary functions can be used to define the behaviour of the function symbol  $f$  using the commuting diagram. The higher-order approach calls for additional distributive properties for representation and abstraction functions on derived sorts.

This simulation principle can be applied to data structure refinements in different ways. In the following we will discuss four data structure transformations supporting the refinement of higher-order algebraic specifications.

## 4.2 Simulating Implementation Function Symbols

For the higher-order case we need not only representation and abstraction functions for the basic sorts, but also for the derived sorts. The representation and abstraction functions must be compatible with tupling and function application.

In the following let  $\mathcal{A}$  be a  $\Sigma_1$ -algebra,  $\mathcal{B}$  a  $\Sigma_2$ -algebra and  $\delta : \Sigma_1 \rightarrow \Sigma_2$  a signature morphism. An  $S_1^{\times, \rightarrow}$ -indexed family  $repr : \mathcal{A} \rightarrow \mathcal{B}$  of representation functions  $repr_s : s^{\mathcal{A}} \rightarrow \delta(s)^{\mathcal{B}}$  is called a *data structure representation* for  $\delta$ , if

$$\begin{aligned} repr_{(s_1, \dots, s_n)}(a_1, \dots, a_n) &= (repr_{s_1}(a_1), \dots, repr_{s_n}(a_n)) \\ repr_{s_2}(f(a)) &= repr_{(s_1 \rightarrow s_2)}(f)(repr_{s_1}(a)) \end{aligned}$$

holds. Similarly, a  $(S_1^{\times, \rightarrow})$ -indexed family  $abstr : \mathcal{B}|_{\delta(\Sigma_1)} \rightarrow \mathcal{A}$  of abstraction functions  $abstr_{\delta(s)} : \delta(s)^{\mathcal{B}} \rightarrow s^{\mathcal{A}}$  is called a *data structure abstraction* for  $\delta$ , if

$$\begin{aligned} abstr_{(\delta(s_1), \dots, \delta(s_n))}(b_1, \dots, b_n) &= (abstr_{\delta(s_1)}(b_1), \dots, abstr_{\delta(s_n)}(b_n)) \\ abstr_{\delta(s_2)}(g(b)) &= abstr_{(\delta(s_1) \rightarrow \delta(s_2))}(g)(abstr_{\delta(s_1)}(b)) \end{aligned}$$

holds. Without loss of generality we assume in the sequel that the signature morphism is injective; otherwise we introduce separate abstraction functions for each sort in  $\delta^{-1}(s)$ . For notational simplicity, we write  $abstr : \mathcal{B} \rightarrow \mathcal{A}$  for abstraction functions.

According to Fig. 1 there are different possibilities of simulating the behaviour of the implementation functions in terms of the original function. In the following we will explore four possibilities and describe how these simulations contribute to a refinement of higher-order algebraic specifications.

**L-Simulation.** The L-simulation uses representation functions only. Here the interpretations of abstract function symbols are mapped to the interpretations of the implementation function symbols:

$$repr_s(f^{\mathcal{A}}) = \delta(f)^{\mathcal{B}} \quad (1)$$

This equation implies that the  $\Sigma_1$ -algebra  $\mathcal{A}$  is a subalgebra of  $\mathcal{B}$  via the signature morphism  $\delta$ .

**Theorem 1 (L-Simulation).** *Let  $\mathcal{A}$  be a  $\Sigma_1$ -algebra,  $\mathcal{B}$  a  $\Sigma_2$ -algebra, and  $repr : \mathcal{A} \rightarrow \mathcal{B}$  a data structure representation for a signature morphism  $\delta : \Sigma_1 \rightarrow \Sigma_2$ . If  $repr$  validates Equation (1) for all  $f \in F_1$ , then  $\mathcal{A} \stackrel{\delta}{\sqsubseteq} \mathcal{B}$  holds.*

The proof exploits the fact that  $repr$  forms a homomorphism from the abstract algebra  $\mathcal{A}$  to a subalgebra of the implementation algebra  $\mathcal{B}$ .

An obvious implication of this theorem reveals how representation functions can be used to describe a refinement of specifications in an abstract way.

**Corollary 1.** *If for a given signature morphism  $\delta : \Sigma_1 \rightarrow \Sigma_2$  and for all  $\mathcal{B} \in CGEN(\Sigma_2, E_2)$  there is  $\mathcal{A} \in CGEN(\Sigma_1, E_1)$  and a data structure representation  $repr : \mathcal{A} \rightarrow \mathcal{B}$  for  $\delta$  such that Equation (1) holds, then we have  $(\Sigma_1, E_1) \xrightarrow{\delta} (\Sigma_2, E_2)$ .*

Heading for mechanizable manipulations of specifications, we will present transformation rules in Section 5 that exploit Corollary 1.

**L<sup>-1</sup>-Simulation.** This kind of simulation makes use of abstraction functions only. Here the abstraction of the implementation function agrees with the interpretation of the original functions:

$$\text{abstr}_{\delta(s)}(\delta(f)^{\mathcal{B}}) = f^{\mathcal{A}} \quad (2)$$

As a consequence the  $\Sigma_1$ -algebra  $\mathcal{A}$  is congruent to  $\mathcal{B}$  via the signature morphism  $\delta$ .

**Theorem 2 (L<sup>-1</sup>-Simulation).** Let  $\mathcal{A}$  be a  $\Sigma_1$ -algebra,  $\mathcal{B}$  a  $\Sigma_2$ -algebra, and  $\text{abstr} : \mathcal{B} \rightarrow \mathcal{A}$  a data structure abstraction for a signature morphism  $\delta : \Sigma_1 \rightarrow \Sigma_2$ . If  $\text{abstr}$  validates Equation (2) for all  $f \in F_1$ , then  $\mathcal{A} \stackrel{\delta}{\approx} \mathcal{B}$  holds.

The proof constructs a  $\Sigma_2$ -congruence on the algebra  $\mathcal{B}$  by relating all elements that are mapped to the same abstract element in the algebra  $\mathcal{A}$ .

This theorem forms the basis for refinements described by data structure abstractions.

**Corollary 2.** If for a given signature morphism  $\delta : \Sigma_1 \rightarrow \Sigma_2$  and for all  $\mathcal{B} \in \text{CGEN}(\Sigma_2, E_2)$  there is  $\mathcal{A} \in \text{CGEN}(\Sigma_1, E_1)$  and a data structure abstraction  $\text{abstr} : \mathcal{B} \rightarrow \mathcal{A}$  for  $\delta$  such that Equation (2) holds, we have  $(\Sigma_1, E_1) \xrightarrow{\delta} (\Sigma_2, E_2)$ .

In Section 5 we will demonstrate how the L<sup>-1</sup> simulation can be used for a mechanizable data structure transformation.

**U-Simulation.** This simulation uses both data structure representation and abstraction. The implementation functions are implicitly characterized by their composition with the representation and abstraction functions. Additionally, for this simulation it must be assured that the composition of abstraction and representation agrees with the identity.

$$f^{\mathcal{A}} = \text{abstr}_{\delta(s_2)} \circ \delta(f)^{\mathcal{B}} \circ \text{repr}_{s_1} \quad (3)$$

$$id_{\delta(s)} = \text{repr}_s \circ \text{abstr}_{\delta(s)} \quad (4)$$

This simulation can be reduced to the L-simulation. Hence, the conditions for a U-simulation imply that  $\mathcal{A}$  is a subalgebra of  $\mathcal{B}$  via the signature morphism  $\delta$ .

**Theorem 3 (U-Simulation).** Let  $\mathcal{A}$  be a  $\Sigma_1$ -algebra,  $\mathcal{B}$  a  $\Sigma_2$ -algebra,  $\text{repr} : \mathcal{A} \rightarrow \mathcal{B}$  a data structure representation for a signature morphism  $\delta : \Sigma_1 \rightarrow \Sigma_2$ , and  $\text{abstr} : \mathcal{B} \rightarrow \mathcal{A}$  a data structure abstraction for  $\delta$ . If  $\text{repr}$  and  $\text{abstr}$  validate Equations (3) and (4) for all  $f \in F_1$ , then  $\mathcal{A} \stackrel{\delta}{\sqsubseteq} \mathcal{B}$  holds.

Similarly to the L-simulation also this theorem offers a possibility for the refinement of specifications.

**Corollary 3.** If for a given signature morphism  $\delta : \Sigma_1 \rightarrow \Sigma_2$  and for all  $\mathcal{B} \in \text{CGEN}(\Sigma_2, E_2)$  there is  $\mathcal{A} \in \text{CGEN}(\Sigma_1, E_1)$ , a data structure representation  $\text{repr} : \mathcal{A} \rightarrow \mathcal{B}$  for  $\delta$ , and a data structure abstraction  $\text{abstr} : \mathcal{B} \rightarrow \mathcal{A}$  for  $\delta$  such that Equations (3) and (4) hold, then we have  $(\Sigma_1, E_1) \xrightarrow{\delta} (\Sigma_2, E_2)$ .

The introduced U-simulation requires a higher effort, since both representation and abstraction function must be provided.

**$U^{-1}$ -Simulation.** This simulation provides an explicit description of the implementation functions using the composition of the abstraction function, the original function, and the representation function. For this simulation it must again be assured that the composition of representation and abstraction agrees with the identity.

$$\delta(f)^{\mathcal{B}} = \text{repr}_{s_2} \circ f^{\mathcal{A}} \circ \text{abstr}_{\delta(s_1)} \quad (5)$$

$$id_s = \text{abstr}_{\delta(s)} \circ \text{repr}_s \quad (6)$$

The  $U^{-1}$ -simulation can be reduced to the  $L^{-1}$ -simulation. As a consequence, the algebra  $\mathcal{A}$  is equivalent to  $\mathcal{B}$  via the signature morphism  $\delta$ .

**Theorem 4 ( $U^{-1}$ -Simulation).** Let  $\mathcal{A}$  be a  $\Sigma_1$ -algebra,  $\mathcal{B}$  a  $\Sigma_2$ -algebra,  $\text{repr} : \mathcal{A} \rightarrow \mathcal{B}$  a data structure representation for a signature morphism  $\delta : \Sigma_1 \rightarrow \Sigma_2$  and  $\text{abstr} : \mathcal{B} \rightarrow \mathcal{A}$  a data structure abstraction for  $\delta$ . If  $\text{repr}$  and  $\text{abstr}$  validate Equations (5) and (6), then  $\mathcal{A} \xrightarrow{\delta} \mathcal{B}$  holds.

This theorem opens a further possibility of refining specifications with data structure representation and abstraction functions.

**Corollary 4.** If for a given signature morphism  $\delta : \Sigma_1 \rightarrow \Sigma_2$  and for all  $\mathcal{B} \in \text{CGEN}(\Sigma_2, E_2)$  there is  $\mathcal{A} \in \text{CGEN}(\Sigma_1, E_1)$ , a data structure representation  $\text{repr} : \mathcal{A} \rightarrow \mathcal{B}$  for  $\delta$  and a data structure abstraction  $\text{abstr} : \mathcal{B} \rightarrow \mathcal{A}$  for  $\delta$  such that Equations (5) and (6) hold, then we have  $(\Sigma_1, E_1) \xrightarrow{\delta} (\Sigma_2, E_2)$ .

This kind of simulation is often preferred for practical purposes, because it defines the implementation function in a unique and explicit way. On the other hand, the  $U^{-1}$ -simulation leaves no freedom in designing the implementation function.

### 4.3 Revisiting the Introductory Examples

**Set by Array.** In the introductory example of Subsection 2.2 we illustrated a data structure refinement by constructor translation. This refinement can be described as an  $L$ -simulation with  $\delta(\text{set}) = \text{array}$ . The function  $\text{repr}_{\text{set}} : \text{set}^{\mathcal{A}} \rightarrow \text{array}^{\mathcal{B}}$  represents finite sets generated by  $\text{emptySet}^{\mathcal{A}}$  and  $\text{addElem}^{\mathcal{A}}$  as arrays in the following way:

$$\begin{aligned} \text{repr}_{\text{set}}(\text{emptySet}^{\mathcal{A}}) &= \text{init}^{\mathcal{B}} \\ \text{repr}_{\text{set}}(\text{addElem}^{\mathcal{A}}(a, s)) &= \text{put}^{\mathcal{B}}(\text{repr}_{\text{set}}(s), a, \text{true}^{\mathcal{B}}) \end{aligned}$$

Hereby we assume that the representation function is the identity on the basic sorts  $\text{nat}$  and  $\text{bool}$ .

**Stack by Array and Pointer.** In a similar way we can define the simulation of  $\text{Stack}$  by  $\text{StackByArray}$  from Subsection 2.3 using an  $L^{-1}$ -simulation. The abstraction function  $\text{abstr}_{\text{stack}} : \text{stack}_{\text{new}}^{\mathcal{B}} \rightarrow \text{stack}^{\mathcal{A}}$  for  $\delta$  with  $\delta(\text{stack}) = (\text{nat}, \text{array})$  retrieves stacks from their array representations in the following way:

$$\begin{aligned} \text{abstr}_{\delta(\text{stack})}(\text{zero}^{\mathcal{B}}, a) &= \text{emptyStack}^{\mathcal{A}} \\ \text{abstr}_{\delta(\text{stack})}(\text{succ}^{\mathcal{B}}(m), a) &= \text{push}^{\mathcal{A}}(\text{lookup}^{\mathcal{B}}(\text{succ}^{\mathcal{B}}(m), a)), \\ &\quad \text{abstr}_{\delta(\text{stack})}(m, a)) \end{aligned}$$

Hereby we assume that the abstraction functions for the basic sorts  $elem$ ,  $nat$ , and  $bool$  are the identity functions.

## 5 Mechanizable Transformation Rules

We present mechanizable transformation rules for data structure transformations operating on specifications which follow the model theoretic framework of the preceding section. We first discuss algebraic implementations and proceed with implicit data structure representations afterwards.

A data structure transformation is described by the principle of *algebraic implementation* in a general way. The representation and abstraction functions are inserted into the specification as new function symbols and the application conditions are added as new axioms. The corollaries of the last section are used in a syntactic manner by incorporating the theory of representation and abstraction functions into the specification itself.

Implicit data structure representations refine a specification without enlarging its signature. These refinement steps can be reduced to L-simulations. The application condition is ensured by the properties of the specification and the particular data structure representation.

In the sequel we assume that the names for the abstraction and representation functions are not used as function symbol in any of the involved specifications.

### 5.1 Algebraic Implementations

The approach of algebraic implementation inserts the symbols for the representation and abstraction functions along with the required axioms into the specification. We will present transformation rules using the L- and  $L^{-1}$ -simulation; the other simulations lead to similar transformations.

**Transformation Rules for Algebraic Implementations.** In the following we expect a specification  $(\Sigma_1, E_1)$ , a disjoint signature  $\Sigma_2$ , and a signature morphism  $\delta : \Sigma_1 \rightarrow \Sigma_2$ . We then construct a new specification that forms a refinement using L-simulation.

**Theorem 5.** Let  $(\Sigma_1, E_1)$  be an algebraic specification,  $\Sigma_2$  a disjoint signature and  $\delta : \Sigma_1 \rightarrow \Sigma_2$  a signature morphism. We generate a specification  $(\Sigma_3 = (S_3, F_3), E_3)$  with  $\Sigma_3 \supseteq \Sigma_1 \cup \Sigma_2$  and

$$(F_3)_{(s \rightarrow \delta(s))} = \{repr_s\} \text{ for all } s \in S_1^{\times, \rightarrow} \\ E_3 = E_1 \cup$$

$$\bigcup_{s_1, \dots, s_n, s \in S_1^{\times, \rightarrow}} \left\{ \begin{array}{l} repr_s(f) = \delta(f) \text{ for all } f \in (F_1)_s, \\ repr_{(s_1, \dots, s_n)}(x_1, \dots, x_n) = (repr_{s_1}(x_1), \dots, repr_{s_n}(x_n)), \\ repr_{(s_1 \rightarrow s_2)}(f)(repr_{s_1}(x)) = repr_{s_2}(f(x)) \text{ for all } f \in (F_1)_{(s_1 \rightarrow s_2)} \end{array} \right\}.$$

Then  $(\Sigma_1, E_1) \xrightarrow{\delta} (\Sigma_3, E_3)$  holds.

Similarly, we can construct a new specification that forms a data structure refinement using  $L^{-1}$ -simulation.

**Theorem 6.** Let  $(\Sigma_1, E_1)$  be an algebraic specification,  $\Sigma_2$  be a disjoint signature, and  $\delta : \Sigma_1 \rightarrow \Sigma_2$  a signature morphism. We generate a specification  $(\Sigma_3 = (S_3, F_3), E_3)$  with  $\Sigma_3 \supseteq \Sigma_1 \cup \Sigma_2$  and

$$(F_3)_{(\delta(s) \rightarrow s)} = \{abstr_{\delta(s)}\} \text{ for all } s \in S_1^{\times, \rightarrow}$$

$$E_3 = E_1 \cup$$

$$\bigcup_{s_1, \dots, s_n, s \in S_1^{\times, \rightarrow}} \left\{ \begin{array}{l} abstr_{\delta(s)}(\delta(f)) = f \text{ for all } f \in (F_1)_s \\ abstr_{(\delta(s_1), \dots, \delta(s_n))}(x_1, \dots, x_n) = \\ \quad (abstr_{\delta(s_1)}(x_1), \dots, abstr_{\delta(s_n)}(x_n)) \\ abstr_{\delta(s_1) \rightarrow \delta(s_2)}(h)(abstr_{\delta(s_1)}(x)) = abstr_{\delta(s_2)}(h(x)) \end{array} \right\}.$$

Then  $(\Sigma_1, E_1) \xrightarrow{\delta} (\Sigma_3, E_3)$  holds.

The inserted equations constitute proof obligations; they may also serve as starting point for deriving an algorithmic implementation of the function symbols.

The new specification  $(\Sigma_3, E_3)$  in both theorems can be minimized by inserting the operations  $repr_s$  resp.  $abstr_{\delta(s)}$  and the corresponding axioms only for those derived sorts  $s$  with  $F_{1,s} \neq \emptyset$ . If the specification  $(\Sigma_1, E_1)$  and the signature  $\Sigma_2$  are finite, then the minimized specification  $(\Sigma_3, E_3)$  is finite as well. In this case the refinement  $(\Sigma_1, E_1) \xrightarrow{\delta} (\Sigma_3, E_3)$  can be mechanized by a transformation system.

Using both representation and abstraction functions for a  $U$ -simulation or a  $U^{-1}$ -simulation leads to a further enlargement of the specification. For the sake of brevity we will omit the transformation rule here. The rule inserts, similar to the  $L$ - and  $L^{-1}$ -algebraic implementation, representation and abstraction function symbols along with the corresponding distributive equations and, additionally, the equations of the  $U^{-1}$ -simulation.

Heading for a direct algorithmic axiomatization of the implementation functions, the  $U^{-1}$ -algebraic implementation is superior to the  $L^{-1}$  approach, because it explicitly defines the new function symbols. In the  $L^{-1}$ -algebraic implementation, the implementation functions are not fixed on the elements in the complement of the embedded sort.

**Stacks by Arrays with Pointers.** We will revisit the introductory example and show that the stack by array implementation corresponds to an  $L^{-1}$ -simulation.

We apply Theorem 6 to the specification *Stack* from Subsection 2.3 and obtain the specification *StackByArray*. We present a simplified form using only an abstraction function for the implementation sort *stack*. We assume that the abstraction functions for the remaining sorts *bool* and *nat* are the identity functions.

```

spec StackByArray = Bool + Nat + ElemArray;
sorts
  elem
  stackold = emptyStackold | pushold(elem, stackold)
  stacknew = (nat, array)
ops
  abstr : (stacknew)stackold
  topnew : (stacknew)elem
  popnew : (stacknew)stacknew
  emptyStacknew : stacknew
  pushnew : (elem, stacknew)stacknew

  error : elem
  topold : (stackold)elem
  popold : (stackold)stackold
vars e : elem, s : stack
axioms
  top : topnew(s) = topold(abstr(s))
  pop : abstr(popnew(s)) = popold(abstr(s))
  emptyStack : abstr(emptyStacknew) = emptyStackold
  push : abstr(pushnew(e, s)) = pushold(e, abstr(s))
  top1 : topold(emptyStackold) = error
  top2 : topold(pushold(e, s)) = e
  pop1 : popold(emptyStackold) = emptyStackold
  pop2 : popold(pushold(e, s)) = s
end

```

After supplying axioms for the definition of the abstraction function further derivation steps lead to an algorithmic realization of stacks using pairs of a natural number and an array.

## 5.2 Implicit Data Structure Representation

The following transformation rules of data structure refinements are based on L-simulation. They use implicit data structure representations i.e. the representation functions are not inserted into the specification. Rather they can be constructed using additional information of the particular refinement.

**Introduction of Generation Constraint.** This transformation converts a loose sort into a constructor sort by changing the constructor affiliation of some operations. The step achieves a first implementable data type definition for loose sorts.

**Theorem 7.** Let  $(\Sigma_1, E_1)$  be an algebraic specification with signature  $\Sigma_1 = (S_1, F_1, C_1)$ . Furthermore let  $SB \subseteq S_1$  be a subset of loose sorts and  $F'_1 \subseteq F_1$  a subfamily of operations with result type in  $SB$ . Then we have  $(\Sigma_1, E_1) \xrightarrow{id} (\Sigma_2, E_2)$ , where  $\Sigma_2 = (S_1, F_1, C_1 \cup F'_1)$  and  $E_2 = E_1$ .

The introduction of a generation constraint induces a changed partition of the basic sorts into loose sorts and constructor sorts.

The constructor introduction forms a proper design decision, since the models of the specification are confined to algebras where the former loose sorts are finitely generated. The choice of the constructors widely influences the further development steps.

**Introducing a Generation Constraint for Sets.** We can apply the constructor introduction to the specification *FinSet* from Subsection 2.2 and achieve with the subfamily  $F'_1 = \{ \text{emptySet}, \text{addElem}, \text{union} \}$  of operations the following specification:

```

spec   FinSet = Bool + Nat ;
sorts      set = emptySet | addElem(nat, set) | union(set, set)
ops       memb : (nat, set)bool
vars     m, n : nat, r, s, t : set
axioms
    unioncomm : union(r, s) = union(s, r)
    unionassoc : union(union(r, s), t) = union(r, union(s, t))
    union1 : union(s, emptySet) = s
    union2 : union(s, addElem(m, t)) = addElem(m, union(s, t))
    memb1 : memb(m, emptySet) = false
    memb2 : memb(m, addElem(n, s)) = or(eqNat(m, n), memb(m, s))
end

```

**Enriching a Constructor System.** The set of constructors can be enriched by inserting function symbols into the constructor system of a specification. The constructor enrichment transformation changes the constructor affiliation of a set of function symbols.

**Theorem 8.** Let  $(\Sigma_1, E_1)$  be an algebraic specification with signature  $\Sigma_1 = (S_1, F_1, C_1)$  and  $F'_1 \subseteq F_1$  be a subfamily of operations with a constructor result sort. Assume that all operations  $f \in F_1 \setminus C_1$  are specified completely by axioms in  $E_1$ . Then we have  $(\Sigma_1, E_1) \rightarrow (\Sigma_2, E_2)$  where  $\Sigma_2 = (S_1, F_1, C_1 \cup F'_1)$  and  $E_2 = E_1$ .

The constructors generate the carriers of the models. Enriching the family of constructors enlarges the carriers, and the embedding of the old carriers into the new carriers must fulfill the subalgebra condition. The refined specification defines super-algebras of the models of the original specification; hence constructor enrichments can be seen as L-simulation steps.

**Reducing a Constructor System.** The signature of a specification can be changed by removing a function symbol from the family of constructors. The constructor reduction drops a subfamily of constructors by changing their constructor affiliation. The function symbols have to remain in the signature in order to keep the interface unchanged.

**Theorem 9.** Let  $(\Sigma_1, E_1)$  be an algebraic specification with signature  $\Sigma_1 = (S_1, F_1, C_1)$  and  $C'_1 \subset C_1$  a proper subfamily of constructors. Then we have  $(\Sigma_1, E_1) \rightarrow (\Sigma_2, E_2)$  where  $\Sigma_2 = (S_1, F_1, C_1 \setminus C'_1)$  and  $E_2 = E_1$ .

This step reduces the number of models in general and therefore forms a design decision. Since the generation constraint of  $(\Sigma_1, E_1)$  is a consequence of the generation constraint of  $(\Sigma_2, E_2)$ , the constructor reduction forms a correct refinement step.

If the generation of the carriers does not depend on the dropped constructor symbol, then the transformation forms an injective embedding and an L-simulation with injective representation functions can be constructed.

**Reducing the Constructor System for Sets.** We refine the specification *FinSet* from Subsection 2.2 by reducing the three-element constructor system of sort *set* to *emptySet* and *addElem*:

```

spec   FinSet = Bool + Nat ;
sorts      set = emptySet | addElem(nat, set)
ops       union : (set, set)set
            memb : (nat, set)bool
vars     m, n : nat, r, s, t : set
axioms
            unioncomm : union(r, s) = union(s, r)
            unionassoc : union(union(r, s), t) = union(r, union(s, t))
            union1 : union(s, emptySet) = s
            union2 : union(s, addElem(m, t)) = addElem(m, union(s, t))
            memb1 : memb(m, emptySet) = false
            memb2 : memb(m, addElem(n, s)) = or(eqNat(m, n), memb(m, s))
end

```

**Constructor Implementation.** This transformation implements constructors by terms of the implementation sort. The implementation is given by a complete set of equations describing how a constructor term of the abstract sort is implemented in the implementation sort.

Let  $\delta : \Sigma_1 \rightarrow \Sigma_2$  be a signature morphism,  $X$  an  $S_1^{\times, \rightarrow}$ -indexed family of variables and  $Y$  an  $S_2^{\times, \rightarrow}$ -indexed family of variables with  $X_s = Y_{\delta(s)}$ . A *term translation*  $\kappa : T(\Sigma_1, X) \rightarrow T(\Sigma_2, Y)$  is an  $S_1^{\times, \rightarrow}$ -indexed family of mappings  $\kappa_s : T_s(\Sigma_1, X) \rightarrow T_{\delta(s)}(\Sigma_2, Y)$  such that

- $Var_{\delta(s)}(\kappa(t)) \subseteq Var_s(t)$ ,
- $\kappa_s(x) = x$  for all  $x \in X_s$ ,
- $\kappa$  is compatible with substitutions  $\sigma : \kappa_s(t\sigma) = \kappa_s(t)(\kappa(\sigma))$ .

A set of equations defining a translation for a subset of constructors can canonically be extended to a term translation.

The transformation rule for a constructor implementation replaces all occurring constructor terms by their implementation term.

**Theorem 10.** Let  $(\Sigma_1, E_1)$  be an algebraic specification with  $\Sigma_1 = (S_1, F_1, C_1)$ ,  $C'_1 \subseteq C_1$  be a subfamily of constructors, and  $\delta : \Sigma_1 \rightarrow \Sigma_2$  a signature morphism with  $C_2 \cap \delta(C'_1) = \emptyset$ . Let  $\{t_c = t'_c \mid c \in C'_1\}$  be a constructor translation and  $\kappa : T(\Sigma_1, X) \rightarrow$

$T(\Sigma_2, Y)$  the corresponding term translation. We construct the specification  $(\Sigma_2, E_2)$  with

$$E_2 = \{\kappa(t_1) = \kappa(t_2) \mid t_1 = t_2 \in E_1\} \cup \{t_c = t'_c \mid c \in C'_1\} .$$

Then we have  $(\Sigma_1, E_1) \xrightarrow{\delta} (\Sigma_2, E_2)$ .

The set of equations describes a data structure representation in an implicit way. In order to guarantee the properties of the data structure representation, the specification must show a certain syntactic form. The required properties can be checked using sufficient syntactic criteria. A transformation system can then validate the properties automatically by analysis algorithms.

**Constructor Implementation for Set by Array.** In the introductory example *FinSet* from Subsection 2.2 we implement the sort *set* by the sort *array* using the constructor translation

$$\begin{aligned} \text{emptySet} &= \text{init} \\ \text{addElem}(m, s) &= \text{put}(s, m, \text{true}) . \end{aligned}$$

The canonical term translation replaces all occurrences of the constructors by their implementation and keeps the remaining term structures. We translate the signature elements by the identity signature morphism.

Clearly, the constructor implementation forms a proper embedding, since there are arrays that do not represent sets of natural numbers. The embedding is not surjective and determines the behaviour of the operations only for arrays that emerged from constructor terms of sort *set*. The restriction of constructor implementations to specifications with completely specified operations is essential in this case.

## 6 Mechanizing the Refinement of Data Structures

The Lübeck Transformation System LTS — a tool for the interactive transformation of algebraic specifications — supports the stepwise refinement of algebraic specifications [11]. We illustrate how the refinement rules are implemented in the LTS. The derivations head for algorithmic specifications which can be compiled into Standard ML code. The transformation rules for the refinement of data structures are invoked by transformation commands with suitable parameters.

### 6.1 Lifecycle of a Specification in LTS

After a specification has been loaded, it resides in the system ready for transformation.

*Analysis* After parsing and context checking, LTS examines the specification's semantic properties known from term rewriting. The results of the analysis guide the programmer in making future design decisions.

*Interaction* The user starts a refinement process by selecting one of the loaded specifications. The system enters the specification mode supporting the refinement of the active specification. After each refinement step, the active specification is analysed to update its properties. Single axioms and terms can be transformed in a fine tuning mode during a subdevelopment.

*Refinement Steps* Simple refinement steps comprise the enrichment of the signature, the addition of axioms, term rewriting, and different types of induction. Complex refinement steps are, among others, the fusion transformation [10] and the refinement of data structures treated in this paper.

When the user finishes the transformation process, the refined specification is inserted into the collection of loaded specifications replacing the original version.

## 6.2 Data Structure Refinement with LTS

LTS provides commands for the data structure transformations described in the previous section. The application conditions are either inserted as proof obligations into the specification or — when using implicit data structure representations — evaluated by the analysis algorithms of LTS using sufficient syntactic criteria. User interaction then completes the refinement by inserting concrete axiomatizations of representation and abstraction functions (in case of an algebraic implementation) and deriving algorithmic axiomatizations of the involved function symbols. Here LTS provides strategies for automatically deducing algorithmic definitions. If a strategy fails, the user can manually complete the derivation introducing additional design decisions or proving suitable propositions.

## 6.3 Applying LTS to an Introductory Example

We will demonstrate how LTS copes with the introductory example refining a stack to a pair of an array and a natural number. After starting the system and loading the specifications *Stack* and *ElemArray*, we first extend the specification *Stack* by arrays:

```
add_import "ElemArray";
```

Then we can apply an  $L^{-1}$ -algebraic implementation using the command

```
refine_DS_Lm1("stack = (nat, array)", "StackByArray").
```

The command's execution results in a new specification comprising the abstraction function for sort *stack*, the implementation functions and the new axioms constraining the implementation functions:

```
SPEC StackByArray
SORTS
H stack_old = empty_old | push_old(elem, stack_old)
    stack = (nat, array)
OPS
H abstr : (stack)stack_old
    top : (stack)elem
    pop : (stack)stack
    empty : stack
    push : (elem, stack)stack
H top_old : (stack_old)elem
H pop_old : (stack_old)stack_old
```

## AXIOMS

```

top: top s_6 = top_old (abstr s_6)
pop: abstr (pop s_7) = pop_old (abstr s_7)
empty: abstr(empty) = empty_old
push: abstr (push(e_9,s_10)) = push_old(e_9,abstr s_10)

top1_old: top_old empty_old = error
top2_old: top_old (push_old(e,s)) = e
pop1_old: ...
END

```

In order to retain the interface of specification *Stack*, the old function symbols are renamed rather than introducing new names for the new function symbols.

The user now can add the axioms for the abstraction function:

```

add_axiom"abstr1: abstr(zero,a) = empty_old";
add_axiom"abstr2: abstr(succ(n),a) =
                  push_old(lookup(succ(n),a),abstr(n,a))";

```

The function symbols *abstr*, *top<sub>old</sub>* and *pop<sub>old</sub>* are hidden since they are only used for the axiomatization of the implementation function symbols and will later be omitted. Hidden functions are marked by H in the system output. Later on the implementation functions corresponding to the constructors *emptyStack<sub>old</sub>* and *push<sub>old</sub>* become operations.

The user can now invoke strategies trying to derive an algorithmic axiomatization of implementation functions. For example, the strategy DSfoldunfold applies a complete case analysis accompanied by rewrite steps in order to achieve an algorithmic axiomatization of the function *top*.

```
DSfoldunfold "top";
```

In some cases manual derivation steps like case analysis, rewrite steps, generalization steps help in reaching an algorithmic axiomatization. After deriving algorithmic axiomatizations of the implementation functions, the abstraction function and the abstract functions are superfluous and can be omitted:

```

drop_function"abstr";
drop_function"pop_old";
drop_function"top_old"; ...

```

Finally, we obtain the following refined specification *StackByArray*:

```

SPEC StackByArray
SORTS
    stack      = (nat,array)
OPS
    top       : (stack)elem
    pop       : (stack)stack
    empty     : stack
    push      : (elem,stack)stack

```

## AXIOMS

```

top_1      : top(zero,a_12) = error
top_2      : top(succ m_13,a_14) = lookup(succ m_13,a_14)
pop_1_1    : pop(zero,a_9) = (zero,init)
pop_2_1    : pop(succ m_10,a_11) = (m_10,a_11)
empty_1    : empty = (zero,init)
push_1     : push(e,(m,a)) = (succ m,put(succ m,e,a))
END

```

When applying strategies the LTS generates new variables on demand e.g. when introducing case analyses. The new variables are build by user declared variables postfixed with a number.

The resulting specification is algorithmic and can be translated into Standard ML code.

## 7 Conclusion

The refinement of imperative programs started with the pioneering work of HOARE [17] and DIJKSTRA [8]. Later on, BACK and WRIGHT [1] introduced a refinement calculus based on the *wp*-calculus. The refinement of data structures employed abstraction and representation functions [17] from the beginning. The model-oriented refinement of data structure was more recently discussed in [7]. Algebraic implementations of first-order specifications have early been introduced by BROY [5] without giving transformation rules.

In this paper we laid the theoretical foundations for the refinement of data structures based on representation and abstraction functions. The approach of using representation and abstraction functions was extended from first-order to higher-order specifications. The refinement relation generalizes the traditional model inclusion; it allows embedding carriers into other carriers and forming new carriers by imposing congruences.

We presented several transformations that can be played back to the introduced theory. The algebraic implementation is a general refinement step incorporating the abstraction and representation functions together with their axioms into the specification. The generality of this approach may lead to an overhead of additional function symbols and axioms but these auxiliary function symbols can be dropped after completion of the development.

The constructor implementation avoids this overhead by omitting function symbols for representation and abstraction functions. In contrast to algebraic implementations, the constructor implementation requires an algorithmic shape from the original specifications. In summary, the constructor implementation forms an easy applicable and mechanizable way of refining data structure. However, the restrictions limit the range of possible applications.

For refining loose sorts, we provided a refinement transformation introducing a generation constraint for a loose sort. Under mild restrictions, loose sorts can be refined by enriching or reducing the family of constructors.

The Lübeck Transformation System provides commands for applying the presented transformation rules of this paper. The system combines the syntactic analysis of al-

gebraic specifications with a powerful transformation engine to provide wide-spanning transformations for the refinement of data structures. Practical experiences showed that the transformation of complex entities like data structures appears to be feasible with LTS and the analysis algorithms assist the progress well.

## References

1. R.-J. Back and J. von Wright. *Refinement Calculus, A Systematic Introduction*. Graduate Texts in Computer Science. Springer, 1998.
2. F.L. Bauer, B. Möller, H. Partsch, and P. Pepper. Formal program construction by transformation – computer-aided, intuition-guided programming. *IEEE Transactions on Software Engineering*, 15:165–180, 1989.
3. M. Bidoit, R. Hennicker, and M. Wirsing. Behavioural and abstractor specifications. *Science of Computer Programming*, 25(2–3):149–186, 1995.
4. M. Bidoit, R. Hennicker, and M. Wirsing. Behavioural and abstractor specifications. *Science of Computer Programming*, 25(2–3):149–186, 1995.
5. M. Broy, B. Möller, and M. Wirsing. Algebraic implementations preserve program correctness. *Science of Computer Programming*, 7:35–53, 1986.
6. R.M. Burstall and J.A. Goguen. Putting theories together to make specifications. In *Proc. 5th Intl. Joint Conference on Artificial Intelligence*, pages 1045–1058, 1977.
7. W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Number 47 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.
8. E.W. Dijkstra. Notes on structured programming. In O. Dahl, E.W. Dijkstra, and C.A.R. Hoare, editors, *Structured Programming*. Academic Press, 1971.
9. W. Dosch and S. Magnussen. Algebraic data structure refinement with the Lübeck Transformation System. In K. Indermark and T. Noll, editors, *Kolloquium Programmiersprachen und Grundlagen der Programmierung, Rurberg, Oktober 2001*, number AIB2001-11 in Aachener Informatik Berichte, pages 7–12. RWTH Aachen, 2001.
10. W. Dosch and S. Magnussen. Computer aided fusion for algebraic program derivation. *Nordic Journal of Computing*, 8(3):279–297, 2001.
11. W. Dosch and S. Magnussen. Lübeck Transformation System: A transformation system for equational higher-order algebraic specifications. In M. Cerioli and G. Reggio, editors, *Recent Trends in Algebraic Development Techniques: 15th International Workshop, WADT 2001, Joint with the CoFI WG Meeting, Genova, Italy, April 1-3, 2001. Selected Papers*, volume 2267 of *LNCS*, pages 85–108. Springer, 2002.
12. H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Einführung in die mathematische Logik*. Spektrum Verlag, 1996.
13. H.-D. Ehrich, M. Gogolla, and U.W. Lippek. *Algebraische Spezifikation abstrakter Datentypen*. Teubner, 1989.
14. H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specifications 1, Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.
15. M. Feather. A survey and classification of some program transformation approaches and techniques. In L.G.L.T. Meertens, editor, *Proceedings TC2 Working Conference on Program Specification and Transformation*, pages 165–195. North Holland, 1987.
16. B.M. Hearn and K. Meinke. ATLAS: A typed language for algebraic specifications. In J. Heering et al, editor, *HOA '93, an International Workshop on Higher-Order Algebra, Logic and Term Rewriting*, volume 816 of *LNCS*, pages 146–168. Springer, 1994.

17. C.A.R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271 – 281, 1972.
18. J. Loeckx, H.-D. Ehrich, and M. Wolf. *Specification of Abstract Data Types*. Wiley & Teubner, 1996.
19. J. Loeckx, H.-D. Ehrich, and M. Wolf. Algebraic specification of abstract data types. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 5, pages 217–316. Oxford Science Publications, 2000.
20. S.J. Magnussen. *Mechanizing the Transformation of Higher-Order Algebraic Specifications for the Development of Software Systems*. Logos Verlag Berlin, 2003.
21. K. Meinke. Universal algebra in higher types. *Theoretical Computer Science*, 100:385–417, 1992.
22. K. Meinke and J.V. Tucker. Universal algebra. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, pages 189–411. Oxford Science Publications, 1992.
23. M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 675–788. Elsevier Science Publishers, 1990.
24. M. Wirsing, H. Partsch, P. Pepper, W. Dosch, and M. Broy. On hierarchies of abstract data types. *Acta Informatica*, 20:1–33, 1983.

# Analyzing Relationships to the Quality Level between CMM and Six Sigma

Ji-Hyub Park<sup>1</sup>, Ki-Won Song<sup>1</sup>, Kyung Whan Lee<sup>1</sup>, and Sun-Myung Hwang<sup>2</sup>

<sup>1</sup> School of Computer Science and Engineering, Chung-Ang University  
`{mux98, from6588, kwlee}@object.cau.ac.kr`

<sup>2</sup> Division of Computer and Communications Engineering, Dae-Jeon University  
`sunhwang@dragon.taejon.ac.kr`

**Abstract.** Software quality management, which leads to improve the performance capability of project, has been studied on the many viewpoints. It is hard work to compare the each process level assessed with CMM, SPICE and Six-Sigma, which are the popular techniques for the industry of software and system management. This paper discusses the relationship between the three process assessments and the derivation of overall mapping table which relates the result levels assessed with SPICE, CMM and Six-Sigma. Additionally, we analyze the mapping-effectiveness, which is the most effectual way after mapping of the levels or before, using the Analytic Hierarchy Process method.

## 1 Introduction

Software quality measurement that is critical for producing a reliable product depended on the ability to define and manage the particular process. Therefore, the standard software process assessments such as CMM(Capability Maturity Model), SPICE(Software Process Improvement, and Capability dEtermination) are used to measure the quality levels of the software reliability and process improvement.

SPICE has been developed as the ISO 15504 TR, which is in the suite of standard for the software process assessment, is divided into 40 processes as following 5 categories [2]. The other standard CMM has 18 KPAs(Key Process Area) and defines M.Q(Maturity Questionnaire) in each KPA [3]. On the contrary, Six-Sigma, which is one of the quality management techniques and utilizes a scientific statistics in order to improve the process of project, is divided into 6 levels [10].

The problem existed here is that there is no universal method for the relationship between each process quality management techniques in aspect of the level determination.

This paper is to estimate Six-Sigma and SPICE levels using that of CMM. To accomplish this, we will improve the process performance by relating the results of level assessment of CMM, SPICE, and Six-Sigma based on the studies [8], [9].

## 2 Background Model

In order to map for each process attributes, we will look into the level decision method of CMM, SPICE, and Six-Sigma.

## 2.1 Data Collection and Calibration

The data used in this research is collected and calibrated by the project profile and assessor portfolio as the following descriptions.

### 2.1.1 Collecting the Defect Data

The operational activities of measurement begin collecting data [1]. The procedures defined for collecting and retaining data need to be integrated into our software processes and made operational.

Collecting data is more than just making the measurements. It consists of the implementing our plans, ensuring our work, and sustaining for the measurement activities of the results. The documentations of our procedure include as follows:

- Identifying the responsible persons and organizations
- Specifying where, when, and how measurements will be done
- Defining the procedure to be used for recording and reporting the results

We collected the data by performing the assessment with SPICE and rating SEI maturity questionnaire for CMM/KPA. The requirements help to ensure that the assessment output is self-consistent and provides the evidence for the substantiation of ratings. The assessment has to be conducted in accordant to the documented process that is capable of meeting the assessment purpose. The assessment process shall contain minimally the following activities such as planning, data collection, data validation, process rating, and reporting.

The qualified assessors are educated and certified with ISO Guide 62 and TR 15504, part 6, and we identify the responsible persons to be rating on the maturity questionnaire. The certified assessors, who has assessed with SPICE (ISO TR 15504, ver.3.3, 1998), are also rated with SEI maturity questionnaires.

The project portfolio shows how they assessed as shown in Table 1.

**Table 1.** Project Portfolio.

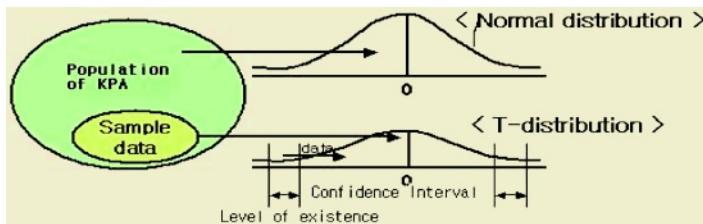
Samples	Development Members	KSLOC	Period of Development(Months)	Number of Assessors
A	302	100	6	10
B	191	150	7	10
C	736	110	9	10
D	1,325	80	8	11
E	1,763	130	10	12

### 2.1.2 Data Calibration

The calibration process of the collected KPA questionnaire data is performed in order to match the SPICE assessment results and verify the reliability of the calibration processes [7], [9].

As shown in Fig. 1, the calibration of KPA items is required to set the sampled data [7] into the rejecting area of the T-distribution.

Handle the KPA Attributes for Data Calibration	
Remove	- Delete process that is out of the range of critical region of distribution in the sample group commonly.
Calibrate	- Calibrate the process, which is out of the range of critical region of distribution, within the confidence interval 99.8% in each sample.

**Fig. 1.** Data Calibration Model.

The calibrated values are derived by re-rating rated values outside the lower and upper rejecting area (out of the confidence interval 99.8%) by interpolation and extrapolation.

The KPA EPML calibrations after and before the sample data compared as presented in Table 2.

**Table 2.** Compare KPA Levels Between Original Data and After Calibration.

Level	Sample	A	B	C	D	E
Original CMM/KPA EPML	3.77	3.38	3.51	3.67	3.19	
CMM/KPA EPML After Calibration	3.23	3.40	3.49	3.51	3.61	

## 2.2 CMM/EPML Calculation Procedure

In order to determine the maturity level of CMM, the key practice questionnaire of KPA formed by 18 categories is to be assessed and rated on the process.

SEI presents the Maturity Questionnaire (M.Q), which corresponds to the key practice questionnaire (1998), in order to support the simplicity in assessment.

For this research we uses Likert's EPML calculation model [4] but uses the maturity questionnaire instead of the rating data of KPA for further details.

$$EPML = 5 \times \left( \sum_{i=1}^n (KPA \%_i) / 100 \right) \times 1/n \quad (1)$$

(*EPML : Estimated Process Maturity Level - Detail process rating on KPA, n : Total process number, i : Count of KPA items.*)

The procedure of tailoring M.Q is shown as follows:

Tailoring Procedure
1. Selecting M.Q by Pareto rule.
2. Rating M.Q on the same target project.
3. Calculate EPML of tailored M.Q rating using Likert's calculation model.

## 2.3 SPICE/EPML Calculation Procedure

We need to calculate SPICE EPML to map between the EPML levels of SPICE and CMM using the calibration model, and the data collected in chapter 2.1.2 [9].

**Table 3.** General Rating Range in SPICE.

	Fully	Largely	Partially	None
Process Achievement(%)	100 ~ 86	85 ~ 51	50 ~ 16	15 ~ 0

SPICE rating is divided into 4 phases as presented in Table 3. However, it causes an ambiguity due to the enumeration and is to make a question as, “Is it Largely (85 ~ 51%), Fully (100 ~ 86%), or Partially (50 ~ 16%) closed?” Therefore, in order to reduce such ambiguity and lead quantitative analysis, we classify the levels as shown in Table 4.

**Table 4.** SPICE Transposition Model Table.

Original	Fully (86~100)			Largely (85~51)			Partially (50~16)			None (15~0)		
Transposition	HF 100	MF 90	LF 88	HF 85	MF 75	LF 60	HF 45	MF 35	LF 20	HF 15	MF 10	LF 5

As presented in Table 5, the results of SPICE levels are calibrated to match the EPML levels of KPA, which show the two similar standard levels of processes.

**Table 5.** Comparison Table between CMM/KPA and SPICE Levels.

Level	Sample	A	B	C	D	E
KPA EPML	3.23	3.40	3.49	3.59	3.61	
SPICE EPML	3.17	3.43	3.47	3.51	3.71	

## 2.4 Six-Sigma Calculation Procedure

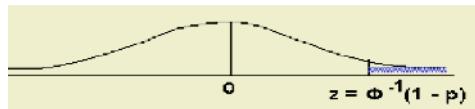
Generally, Six-Sigma level is determined by the numbers of defects within the evaluation range(numbers of defects for the work product). In other words, Six-Sigma level can be determined by the numbers of defect occurred during the process of documenting requirement manifests, analysis & design documents, source code, and quality assurance certificates. SPICE and CMM assessment extracts the numbers of defects occurred in each process to determine the maturity level of the target process.

### 2.4.1 Calculation Type

Calculation types calculating the KPA's and S/W sigma level are shown below:

- **Category Type:** It apply to degree of achievement by questionnaire depicted in Sigma level.

In this paper, we determine Six-Sigma level by defining the numbers of defects occurred during the M.Q selection process for process yield as explained in chapter 2.1.2. In order to decide the Six-Sigma level, determine Sigma value Z (standard normal distribution) used by defect rates as illustrated in Fig. 2 [10].



**Fig. 2.** A Standard Normal Distribution for Determinate Six-Sigma Level.

The following detailed expression explains Fig. 2, and in case of the long-term data is selected then adds 1.5 to the calculated value Z.

- Defect Rate : Number of Defect / Number of Total Process
  - Process First Pass Yield( $p$ ) = 1-Defect Rate
  - $Z = \Phi^{-1}(1 - p)$
  - Sigma Level =  $z + 1.5$
- (2)

( $p$  : Process First-Pass Yield,  $\Phi$  : Accumulation Probability Function of Standard Normal Distribution,  $Z$  : Standard Normal Distribution )

- **Discrete Type:** Data collecting by counting the number of defects. To yield sigma level according to design, calculation must be done through application of the number of DPU.

In case of draw the sigma level of design part, it must be calculated by the Defect Per Unit(DPU).

- DPU : Number of Defect/Number of Total Process
  - Process First Pass Yield( $p$ ) =  $e^{-DPU}$
  - $z = \Phi^{-1}(p)$
  - Sigma Level =  $z + 1.5$
- (3)

( $DPU$  : Defect Per Unit,  $p$  : Process First-Pass Yield,  $e$  : Exponential Function,  $\Phi$  : Accumulation Probability Function of Standard Normal Distribution,  $Z$  : Standard Normal Distribution)

## 2.5 Least Squares Curve Fitting

The objective for applying the *method of least squares* is to find the equation for a curve which best represents a set of data. This involves[14]:

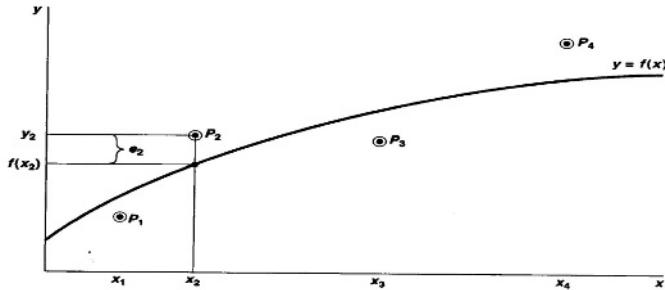
- Selecting an appropriate form for the curve or equation
- Determining the “best” coefficients for the equation

Given the set of data,

$$P_i = (x_i, y_i) \quad i = 1, n \quad (4)$$

Select an appropriate equation to represent the data,

$$y = f(x, a, b, \dots) \quad (5)$$



**Fig. 3.** Least Squares Curve Fitting.

Where  $a, b, \dots$  are coefficients that may be selected for the equation, e.g., for a linear curve fit,

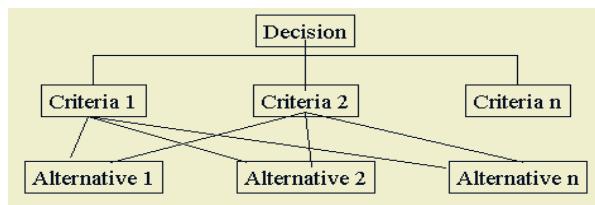
$$y = bx + a \quad (6)$$

The coefficients  $a$  and  $b$  are chosen to give the “best” representation of the data. For the method of least squares, the definition of “best” is the choice that minimizes the sum of the deviations squared.

$$\begin{aligned} e_i &= y_i - f(x_i, a, b, \dots) \quad i = 1, n \\ S &= \sum_{i=1}^n e_i^2 \end{aligned} \quad (7)$$

## 2.6 AHP Method for MCDM (Multi-criteria Decision Making)

AHP (Analytic Hierarchy Process) divides the whole phase of decision-making by several phases and analyzes these phases to lead for the ultimate decision-making. It has ‘Criteria’ under ‘Decision’ (Goal) and ‘Alternatives’ are the bottom of the hierarchy [5].



**Fig. 4.** AHP Structure.

Using the importance intensity as shown in Table 6, calculate the weights and achievements of numerous decision-making criteria by simple pair-wise comparison matrix (1:1 comparison).

**Table 6.** Intensity of Importance.

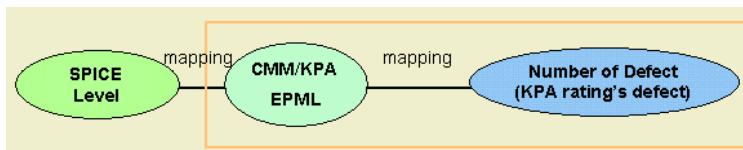
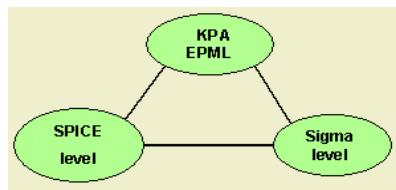
	<b>Definition</b>	<b>Explanation</b>
<b>1</b>	Equal importance	Two activities contribute equally to the objective
<b>3</b>	Weak importance	Experience and judgment slightly favor one over another activity over another
<b>5</b>	Essential or strong	Experience and judgment strongly favor one importance activity over another
<b>7</b>	Very strong or demonstrated	An activity is favored very strongly over another; its dominance demonstrated in practice
<b>9</b>	Absolute importance	The evidence favoring one activity over another is of the highest possible order of affirmation

The advantage of utilizing such AHP analysis supports the rational group-decision results by means of the application of the preferences in each of criteria.

### 3 Mapping for the Level of CMM, SPICE, and Six-Sigma

While the process level determination sited above is performed in individual, and is required to the high costs dues to the undefined relationship for each level. Therefore, it is possible to estimate Six-Sigma level by using the KPA levels when KPA and Six-Sigma level are correlated [9].

It is also possible to estimate the mutuality levels among CMM, SPICE, Six-Sigma by the earlier samples of KPA and SPICE level mapping. Fig. 5 shows the procedures for mapping each level.

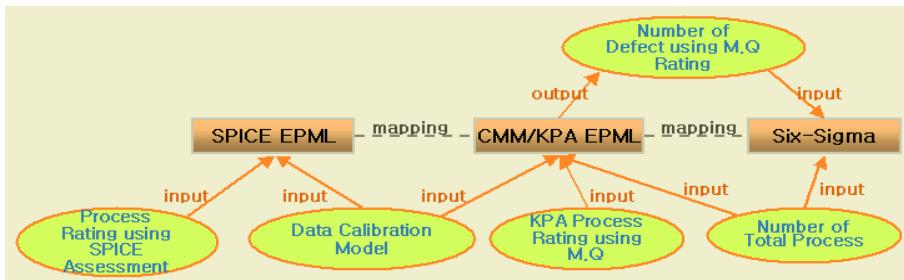
**Fig. 5.** Mapping Procedures.**Fig. 6.** Relationship among the Three Techniques.

Since Six-Sigma level is determined by the process yield, which is calculated by the numbers of defects in the M.Q, it is possible to determine Six-Sigma level by calculating the numbers of defects through KPA EPML level. Upon completion of this mapping phase, we can reason the relationship for each level as shown in Fig. 6.

### 3.1 Input Requirements and Procedure for Mapping Levels

The selected assessors collect the rated data for the objective project of assessment and analyze the confidence that the data, which is out of the range of confidence interval, is considered as the defects in this paper [9]. The input requirements of mapping for each assessment are as follows:

- Numbers of Defects  
: Numbers of defects for M.Q in CMM/KPA
- KPA and SPICE Level Mapping Data  
: Application of KPA EPML and SPICE level mapping data (5 sample data)
- Total Numbers of M.Q in CMM/KPA



**Fig. 7.** Input Elements and Relationships among the Three Techniques.

Fig. 7 shows the input requirements for mapping levels between KPA and Six-Sigma. The determined KPA EPML, rating points, and applicable numbers of M.Q are required. MQ with the rating points deviating from the mean EPML can be defined as defects since it depicts the ambiguity of the process.

The Requirements for Six-Sigma level are the numbers of MQ and process yield derived from the defects by KPA EPML deviation.

### 3.2 Case Study of the Mapping Levels for CMM, SPICE, and Six-Sigma

In this section, we will map and analyze the level between SPICE and KPA using the process assessment results of the 5 samples [4]. First of all, we calculate the Six-Sigma level based on the numbers of defects for each of MQ sample. Then, we calculate the process yield after determining the numbers of defects by KPA EPML level. By using these data, the gap for the levels of SPICE, KPA, and Six-Sigma are analyzed.

#### 3.2.1 Assessment Results of CMM, SPICE, and Six-Sigma

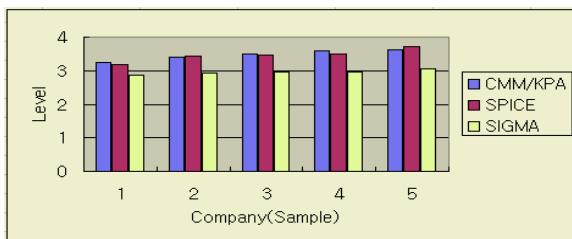
Table 7 shows the results of KPA, SPICE, and Six-Sigma levels for the 5 companies(sample A ~ E).

The numbers of defects used for calculating the Six-Sigma level is presented from the deviations for each KPA level.

As shown in Table 7, Six-Sigma shows the lower level by average of 0.5 (point five) than KPA and SPICE levels, and it is illustrated as Fig. 8.

**Table 7.** Mapping Table with 5 Samples' Assessment Results.

Company	Collected Data		Assessed Level		
	# Total Process	# Defect	CMM/KPA	SPICE	Six-Sigma
A	84	7	3.23	3.17	2.88
B	53	4	3.40	3.43	2.93
C	90	6	3.49	3.47	2.95
D	91	5	3.59	3.51	2.97
E	97	6	3.61	3.71	3.04
<b>Avg.</b>	<b>82.80</b>	<b>5.60</b>	<b>3.46</b>	<b>3.46</b>	<b>2.96</b>

**Fig. 8.** Level Comparison Graph about Each 5 Samples.

### 3.2.2 Mapping Levels between CMM

#### and Six-Sigma Using Least Squares Fitting Method

We need to calculate the process yield by defining the gaps of certain Sigma level in order to map between KPA EPML level 5 and Six-Sigma level 6 based on current data. The process yield in each Six-Sigma level can be calculated by using the gaps of both KPA EPML and the defects by least square fitting method. For this, we needs the least square fitting method for CMM and SPICE level matching.

Therefore, in this chapter, we use least squares fitting to connect each samples to complement the above method.

In order to generate KPA EPML to Six-Sigma level mapping table, we define the precondition, as the estimated mapping table (see Table 9) and apply other sample data.

The preconditions for generating a mapping table are as follows.

Precondition	
<b>Lowest Level</b>	<ul style="list-style-type: none"> <li>Term of Data Collection : Long Term</li> <li>CMM/KPA Level : 1</li> <li>Sigma Level : 1.5 (Long-Term)</li> <li>Process Yield : 50 %</li> </ul>
<b>Top Level</b>	<ul style="list-style-type: none"> <li>Top Level of KPA EPML : 5</li> <li>Top Level of Six-Sigma : 6</li> <li>Process Yield : 99.99965%</li> <li>Number of Defect : 0.0003</li> </ul>

In the presented table, when the numbers of defect is 0.0003, the process yield is 99.99965% as corresponding to Six-Sigma level 6 (required process yield is 99.99966%). Using the mentioned conditions, the process yield and sigma level can be calculated by assessed results of Table 7 and the formula of Least Squares Fitting as follows.

We take least square fitting approximation by polynomials:

$$P_i = (x_i, y_i) \quad i = 1, n \quad (8)$$

$$y = bx + a$$

Given the set of data:

- Given : (x, y), n=2  
(x, y) = (1.00, 1.5), (3.23, 2.88), (3.40, 2.93), (3.49, 2.95), (3.59, 2.97), (3.61, 3.04), (6.00, 6.00)

We can depict this into the following expressions :

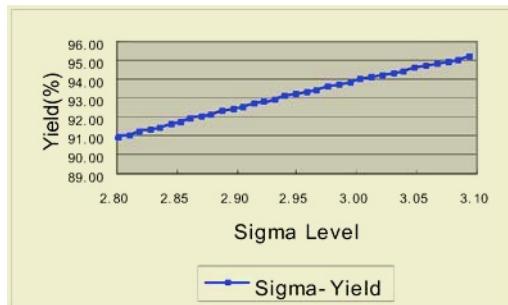
$$y = 0.123x^2 + 0.034x + 1.352 \quad (9)$$

Below table shows the estimated table using above method.

**Table 8.** KPA-Sigma Estimated Table.

Estimated		
CMM/KPA	Long-Term Process Yield(%)	Sigma
3.22	89.8619	2.74
3.23	90.0100	2.75
3.24	90.1573	2.76
:	:	:
3.4	92.4035	2.90
3.41	92.5370	2.90
:	:	:
3.49	93.5759	2.97
3.5	93.7021	2.98
:	:	:
3.59	94.8016	3.07
3.6	94.9197	3.07
:	:	:
3.61	95.0370	3.08
3.62	95.1535	3.09
:	:	:

Shaded in above table, it represents a point of 5 sample's level. Below graph represent the above table which shows the upward process yield in order to sigma level.

**Fig. 9.** Sigma Level and Process Yield.

The differences of Six-Sigma level, while other samples data are applied to the estimated level, are presented in Table 9. In addition, it shows the SPICE levels corresponded to CMM/KPA by the existing SPICE and CMM/KPA mapping research.

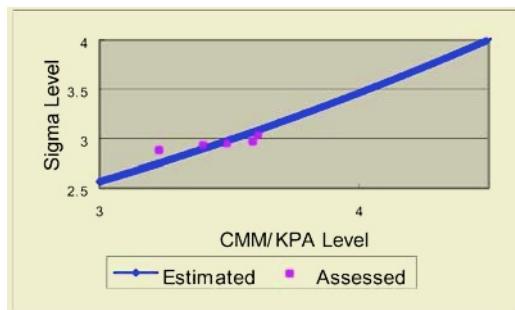
**Table 9.** Six-Sigma Level Gaps between Estimated and Assessed Samples.

CMM/KPA	Estimated		Assessed	
	Long-Term Process Yield(%)	Sigma	Sigma	SPICE
3.22	89.8619	2.74		
3.23	90.0100	2.75	2.88(A)	3.17(A)
:	:	:		
3.38	92.1340	2.88		
:	:	:		
3.4	92.4035	2.90	2.93(B)	3.49(B)
:	:	:		
3.44	92.9326	2.93		
3.45	93.0629	2.94		
3.46	93.1923	2.95		
:	:	:		
3.49	93.5759	2.97	2.95(C)	3.47(C)
:	:	:		
3.56	94.4424	3.04		
:	:	:		
3.59	94.8016	3.07	2.97(D)	3.51(D)
:	:	:		
3.61	95.0370	3.08	3.04(E)	3.71(E)
:	:	:		

- **Estimated Level:** Mapping level relationship between KPA and Six-Sigma based on Table 8
- **Assessed Level:** Mapping level relationship between KPA and Six-Sigma using actual sample(A~E) data

Table 9 shows the estimated and assessed sample sigma level gap of each KPA level is average 0.64 and the highest gap is sample A(0.13).

We get graph as Fig. 10.

**Fig. 10.** KPA's Sigma Level of 5 sample(A~E).

### 3.3 S/W Sigma Level

The above method mapped by yielding KPA's sigma level. In this chapter, we will calculate S/W's sigma level and apply to estimated level in chapter 3.2.

#### 3.3.1 Input Data

For the S/W Sigma level, we define the defect of software as follows:

$$\text{Issue} < \text{Problem} < \text{Defect} < \text{Risk}$$

- Issue: All kinds of issue which is lead to problem
- Problem: Some of issues which give rise to a defect
- Defect: The problem which must be remove or change for the S/W completeness
- Risk: The defect which lead to change of quality, cost or schedule.

#### [ Defect Instance ]

- Specification error or incompleteness (including misspelling)
- Inconsistent with standard specification of requirement analysis, design and coding
- Code error
- Error of data expression
- Inconsistency of module interface
- Test process or test case error

In order to calculate S/W's sigma level, we define the defect as follows and describe the defect measurement results in Table 10.

**Table 10.** Input Data For S/W Sigma Level.

Requirement Analysis		Design				Coding		
Item	Count		Item	Count		Item	Count	
	Project A	Project B		Project A	Project B		Project A	Project B
Init. Req	173	24	Total Req. Function	330	45	Total SLOC	311300	19149
Change Req	56	1	Change Req. Function	14	69	Defect	212	28
Add Req	21	0	Defect	306	69			
Total Req	194	24						

- *Init. Requirement*: The number of initial requirement item, excluded the change or add the requirement items.
- *Total Requirement*: The number of final requirement items, which is included initial, change or add requirement items.
- *Defect*: reference above defect instance ( In case of requirement analysis, not add requirement but change requirement item is defined as defect. )

If the number of defect is 0 in requirement analysis, we should consider as potential defect(in case of non-detection of defect, etc.). Therefore, if the number of defect in design part is high, we define the potential number of defect as 1(at least),

### 3.3.2 Applying the Estimated Level to S/W Sigma

Based on above input data, we calculate the Sigma level below table.

**Table 11.** S/W Sigma Level.

Project		# Total Process	# Defect	Defect Rates	First-Yield	Sigma Level
P1	Req. Analysis	194	56	0.2887	0.7113	1.76
	Design	330	306	0.9273	0.3956	2.76
	Coding	311300	212	0.0007	0.9993	5.5
P2	Req. Analysis	24	1	0.0417	0.9583	3.23
	Design	45	69	1.5333	0.2158	2.07
	Coding	19149	28	0.0015	0.9985	4.46

In above table, [Exp.2] in chapter 2.4.1 is applied for requirement analysis and coding, but [Exp. 3] is applied in design because DPU must be considered.

**Table 12.** Overall Sigma Level Apply with Weight.

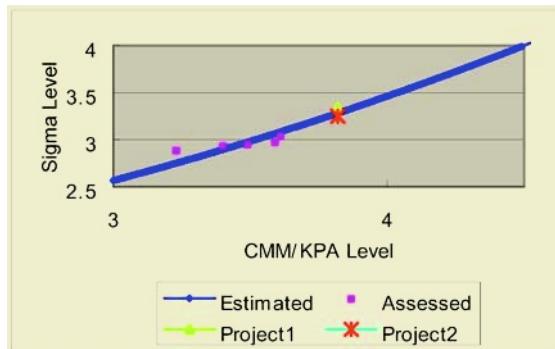
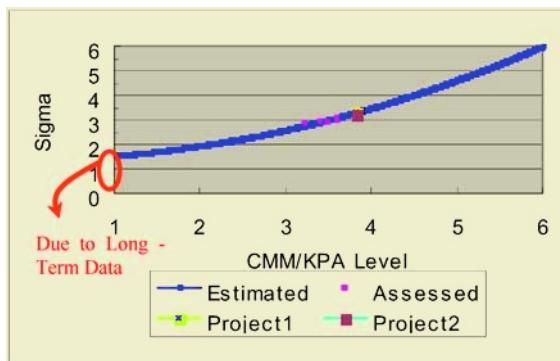
Project		Sigma Level	Overall Sigma Level	CMM/KPA	
P1	Req. Analysis	1.76	3.34	3.82	
	Design	2.76			
	Coding	5.5			
P2	Req. Analysis	3.23	3.25		
	Design	2.07			
	Coding	4.46			

In above table, overall Sigma Level is calculated by the expression as below [Exp. 9].

$$\text{Overall Sigma level} = \sum_{i=1}^n IS / n \quad (10)$$

( IS : Individual Sigma Level, n: number of parts(ex. design or coding. etc ))

By applying the above sigma level to the estimated level Table 8,9 in chapter 3.2, we yield the following result of estimated levels using least squares fitting and assessed levels of 5 KPA's sample.

**Fig. 11.** KPA(A~E) and Six Sigma(Project 1,2) Level.**Fig. 12.** Overall Estimated Level of KPA and Six Sigma.

As shown above, gap between the estimated and assessed level is within 0.13 which follows the CMM/KPA sigma level tendency in chapter 3. Also, in case of long-term data, it must be add 1.5 to calculated sigma level. Therefore, if KPA's level is 1, Sigma level must be starting at least 1.5.

Table 13 shows the estimated level ranges table between Sigma and KPA using estimated level as Table 9.

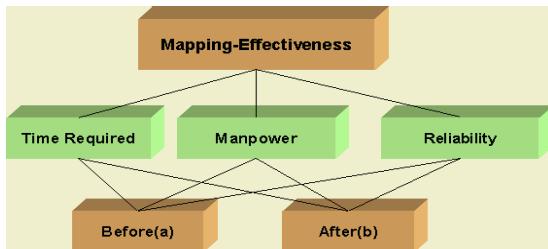
**Table 13.** Sigma Level Ranges by KPA Level.

CMM	Sigma
1	1.5 ~ 1.91
2	1.91 ~ 2.55
3	2.5 ~ 3.45
4	3.46 ~ 4.59
5	4.6 ~ 5.99

We assumption that the data is long term, therefore when the KPA level is 1, Sigma level should be more over 1.5.

## 4 Mapping-Effectiveness Analysis Using AHP

Using AHP, we will discuss which is the more effective way between the performing of each three techniques individually(mapping before) and mapping after that drives the process levels of each three techniques by the level assessed results caused from one of these techniques according to the previous mapping researches.



**Fig. 13.** AHP Structure about Mapping-Effectiveness.

The criteria applied to expense are the time-required, manpower-commitment, and reliability of mapping levels. Alternatives are selected by the basis for mapping before (a) and after (b). This calculation by AHP is as [Exp.11].

- **Achievement Rate** (11)  
 $= \sum(\text{Individual Criteria}/\text{Sum of Criteria}) * 1/\text{Numbers of Criteria}$
- **Weight** Calculation Phase  
 $= \sum(\text{Individual Alternative}/\text{Sum of Alternative}) * 1/\text{Numbers of Alternative}$
- **Results (calculation)**  
 $= \text{Alternative } a: \text{Achievement rate} + \text{Weight}(a)$   
 $= \text{Alternative } b: \text{Achievement rate} + \text{Weight}(b)$

The analyzing results of the decision-making criteria for mapping-effectiveness determined by the previous methods are presented in Table 14.

**Table 14.** Decision-Making Criteria.

	Time Required	Manpower	Reliability	Achievement Rate
<b>Time Required</b>	1	1	0,333	0,200
<b>Manpower</b>	1	1	0,333	0,200
<b>Reliability</b>	3	3	1	0,600
<b>Sum</b>	5	5	1,667	1,000

The time required and manpower commitment is a side of cost factors, which they have equal importance for each other, and the reliability (the reliability of each level after mapping levels) is more important than these.

Table 15 expresses the effectiveness of the degree of mapping levels between before (a) and after (b).

**Table 15.** Importance Rate of Time Required.

	a	b	weight
a	1	0.125	0.111
b	8	1	0.889
sum	9	1.125	1.000

**Table 16.** Importance of Manpower Commitment.

	a	b	weight
a	1	0.125	0.111
b	8	1	0.889
sum	9	1.125	1.000

**Table 17.** Importance of Reliability.

	a	b	weight
a	1	3	0.750
b	0.333	1	0.250
sum	1.333	4	1.000

As presented in Table 16, the mapping before (a) performed the levels individually are more expensive by three times than that of (b) as for the time required and manpower. Therefore, the item (a) is more important than the item (b).

As you see the reliability, all the elements are in the actual level in (a) so that they have strong reliability. Otherwise, in case of the mapping after (b), the correct element is the only one (actually performed element) and the remained elements have some deviations. According to this, the important rate of (a) is higher than (b) in the reliability, but their deviations are small as the previously represented in chapter 3, and then we will give the less important rate than the time required and manpower.

Using the degree of achievement and weight from Table 15~17, we are able to determine the points from each alternative as presented in Table 18.

**Table 18.** Alternative Point using Achievement and Weight.

Criterion	Time Required	Manpower	Reliability	Calculation (Achievement + Weight)
Achievement Rate(A)	0.200	0.200	0.600	
weight(a)	0.111	0.111	0.750	0.494
weight(b)	0.889	0.889	0.250	0.506

The degree of achievement and weight is presented in Table 18. The final point for alternative selection is calculated by the adding degree of weight (a) and (b) to the degree of achievement (A). According to the table presented, we can see that the final point for (a) (before mapping) is 0.494, and (b) (after mapping) is 0.506 that makes an alternative (b) for the mapping effective solution.

## 5 Conclusions

We compare and analyze the level similarities between the three popular software process management techniques and also drive the overall levels of mapping table.

First of all, the rating questionnaires selected by the Pareto rule for the questionnaire items of KPA for EPML is calculated. In addition using the defects numbers in questionnaire checking to correspond with the 5 steps level grade of CMM and 6 steps level grade of Six-Sigma, we are able to calculate the process yield and drew process yield of CMM level. That is, the mapping is performed by means of comparing with the Six Sigma levels according to the yield value calculated by EMPL that is the evaluation of CMM/KPA and number of defects. Also, according to the confirmation level equivalence of CMM and SPICE, the mapping of SPICE and 6 Sigma levels become available in the reference papers, and the mapping research between these process levels, which shows an effect for reducing the expense and period to decide each level, are proved by the mapping-effectiveness analysis using AHP analysis method.

Takes the advantages of results, we can analyze SPI activity by SPICE and CMM using the GQM method to measure somewhat the goal is achieved. In order for this goal, the 6 Sigma levels are extracted by calculating the process yield by means of the same method for the defects derived from the process of the reviews of SW development and independent validation & verification. For the mapping of SPICE, CMM, and Six Sigma levels using this SW development process, it can be utilized to the Experience Management System(EMS) construction that is used for accumulating the project experience data.

## References

1. Kyung Whan Lee, "Modeling for High Depending Computing", Keynote speech, 5th KISS SIG SE, The Korea Conference on Software Engineering Book, phenix park, Feb, 20-21, 2003
2. ISO/IEC JTC1/SC7 15504, "Information Technology Software Process Assessment", 1998.01
3. Geir Amsjo, "Capability Maturity Model(CMM) for Software Development", <http://www.ifi.uio.no/in331/foiler/SW-CMM.pdf>
4. Barry W. Boehm, etc , "Software cost estimation with COCOMOII" , Prentice Hall , 2000
5. Ramayya, "Multi Criteria Decision Making", <http://expertchoice.co.kr/Expert%20Choice/Methodology/mnuMeth-od.htm>
6. KSPICE (Korea Association of Software process Assessors), SPICE "Assessment Report" <http://kaspa.org>, 2002
7. Kyung-Whan Lee , "Research for HDC Modeling", Proceedings of the 5<sup>th</sup> Korean Conference on Software Engineering , 2003.02.20 – 22
8. Woo-Song Kim , "Reliability Test of Maturity Questionnaire Selection Model Through KPA Rating Data Calibration", Conference proceedings, KISS, Jeju, Korea, Apr, 25-26
9. Ki-Won Song , "Research about confidence verification of KPA question item through SEI Maturity Questionnaire's calibration and SPICE Level metathesis modeling", SERA'03, San Francisco, 2003.06
10. Peter S. Pande , "The Six Sigma Way Team Fieldbook", McGraw-Hill, December, 2001
11. "Computer Applications of Numerical Methods", Bosung Moonhwasa, 1977
12. *Lecture 14 - Interpolation Methods*, June 29, 2001, <http://stommel.tamu.edu/~esandt/Teach/Summer01/CVEN302>
13. Holistic Numerical Methods institute, "Newton's Divided Difference Polynomial Method", <http://numericalmethods.eng.usf.edu>
14. Least Squares Curve Fitting <http://www.me.ttu.edu/faculty/oler/me1315/website/notes/Least%20Squares%20Curve%20Fitting.ppt>

# Deterministic End-to-End Guarantees for Real-Time Applications in a DiffServ-MPLS Domain

Steven Martin<sup>1</sup>, Pascale Minet<sup>2</sup>, and Laurent George<sup>3</sup>

<sup>1</sup> Ecole Centrale d'Electronique, LACCSC, 53 rue de Grenelle, 75007 Paris, France  
[steven.martin@ece.fr](mailto:steven.martin@ece.fr)

<sup>2</sup> INRIA, Domaine de Voluceau, Rocquencourt, 78153 Le Chesnay, France  
[pascalle.minet@inria.fr](mailto:pascalle.minet@inria.fr)

<sup>3</sup> Université Paris 12, LIA, 120, rue Paul Armangot, 94400 Vitry, France  
[george@univ-paris12.fr](mailto:george@univ-paris12.fr)

**Abstract.** In this paper, we are interested in providing deterministic end-to-end guarantees to real-time applications in the Internet. We focus on two QoS parameters: the end-to-end response time and the end-to-end jitter, parameters of the utmost importance for such applications. We propose a solution, very simple to deploy, based on a combination of DiffServ and MPLS. The *Expedited Forwarding* (EF) class of the *Differentiated Services* (DiffServ) model is well adapted for real-time applications as it is designed for flows with end-to-end real-time constraints. Moreover *MultiProtocol Label Switching* (MPLS), when applied in a DiffServ architecture, is an efficient solution for providing *Quality of Service* (QoS) routing. The results of our worst case analysis enable to derive a simple admission control for the EF class. Resources provisioned for the EF class but not used by this class are available for the other classes.

**Keywords:** DiffServ, MPLS, EF class, QoS, real-time constraints, deterministic guarantee, admission control, worst case end-to-end response time.

## 1 Introduction

New applications, particularly real-time applications (including voice and/or video), make the best-effort service model inadequate. Indeed, this model cannot provide Quality of Service (QoS) guarantees, in terms of bandwidth, delay, jitter or packet loss. To answer the end-to-end QoS requirements in networks, the IETF has first defined the *Integrated Services* (IntServ) architecture [7]. This service model involves per flow signalling. Each router has to process signalling messages and to maintain a soft state per flow. The low scalability of the IntServ architecture led the IETF to define the *Differentiated Services* (DiffServ) architecture [6]. This architecture is based on traffic aggregation in a limited number of classes. In particular, the *Expedited Forwarding* (EF) class has been proposed for applications requiring low end-to-end packet delay, low delay jitter and low packet loss (e.g. voice and video applications that are delay and jitter sensitive), the EF class being the highest priority class.

In addition, Internet Service Providers need traffic engineering to optimize the utilization of network resources and to improve performance of highly demanding traffics.

*MultiProtocol Label Switching* (MPLS) has been introduced to improve routing performance. This technology is strategically significant for traffic engineering [3], especially when it is used with constraint-based routing. It requires route optimization based on a set of constraints, and route assignment. The former can be done by extending current routing protocols, like OSPF [1]; the latter has to perform label distribution and support explicit routing, like RSVP-TE [2]. As MPLS allows to fix the path followed by all packets of a flow, it makes easier the QoS guarantee. Moreover, a MPLS label can easily represent a DiffServ class of service.

In this paper, we propose a solution to guarantee deterministic end-to-end response times and jitters to all the EF flows in a DiffServ domain. Indeed, beyond the qualitative definition of the offered QoS, no deterministic end-to-end guarantees have been proved for the EF class. On the other hand, we show how the label switching technology can be used to implement our solution.

The rest of the paper is organized as follows. In section 2, we define the problem and the different models. Section 3 briefly discusses related work. The solution, given in section 4, is based on a combination of DiffServ and MPLS. Then, we show in section 5 how to compute upper bounds on the end-to-end response time and the end-to-end delay jitter of any flow belonging to the EF class, based on a worst case analysis. We then derive from this analysis a simple admission control defined in section 6. In section 7, we show how close the bounds we have found are to the real worst case end-to-end response times and jitters. For that purpose, we have designed a simulation tool that does an exhaustive analysis. Comparison results are illustrated by an example. Finally, we conclude the paper in section 8.

## 2 The Problem

We investigate the problem of providing deterministic QoS guarantees, in terms of end-to-end response time and delay jitter, to any flow belonging to the *Expedited Forwarding* class in a DiffServ domain. The end-to-end response time and delay jitter of an EF flow are defined between the ingress node and the egress node of the flow in the considered domain. In this paper, we want to provide the guarantee that the end-to-end response time (resp. the delay jitter) of any flow belonging to the EF class will not exceed  $D$  (resp.  $J$ ), where  $D$  and  $J$  are two parameters of the considered DiffServ domain. As we make no particular assumption concerning the arrival times of packets in the domain, the feasibility of a set of flows belonging to the EF class is equivalent to meet both constraints, whatever the arrival times of the packets in the domain.

Moreover, we assume the following models.

### 2.1 DiffServ Model

In the DiffServ architecture [6], traffic is distributed over a small number of classes. Packets carry the code of their class. This code is then used in each DiffServ-compliant router to select predefined packet handling functions (in terms of queuing, scheduling and buffer acceptance), called *Per-Hop Behavior* (PHB). Thus, the network is divided in two parts: the nodes at the boundary of the network (ingress and egress routers),

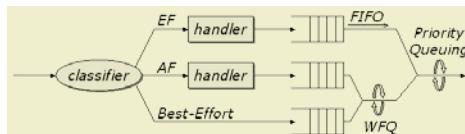
that perform complex treatments (packet classification and traffic conditioning) and the nodes in the core network (core routers), that forward packets according to their class code.

DiffServ provides coarser levels of service than IntServ due to flow aggregation, but is implementable in large networks, simplifies the construction of multi-domain services, and is well adapted for tariffing by class of service. Several per-hop behaviors have been defined:

- the *Assured Forwarding* (AF) PHB group [14]. Four classes, providing more or less resources in terms of bandwidth and buffers, are defined in the AF service. Each class manages three different drop priorities representing the relative importance of a packet in the class.
- the *Expedited Forwarding* (EF) PHB [15]. Traffic belonging to the EF service is delivered with very low latency and drop probability, up to a negotiated rate. This service can be used for instance by IP telephony.
- the *Best-Effort Forwarding* PHB is the default one.

## 2.2 Scheduling Model

We consider that a DiffServ-compliant router implements best-Effort, AF and EF classes. When a packet enters the node scheduler, it is scheduled with the other packets of its class waiting for processing. We consider that the scheduling of the EF class is FIFO. Moreover, we assume that packet scheduling is non-preemptive. Therefore, the node scheduler waits for the completion of the current packet transmission (if any) before selecting the next packet.



**Fig. 1.** DiffServ-compliant router

As illustrated by figure 1, the EF class is scheduled with a *Fixed Priority Queuing* scheme with regard to the other classes. Thus, the EF class is served as long as it is not empty. Packets in the best-Effort and AF classes are served according to *Weighted Fair Queuing* (WFQ) [18]. In this way, EF traffic will obtain low delay thanks to *Fixed Priority Queuing* scheduler and AF traffic will receive a higher bandwidth fraction than best-Effort thanks to wfq. Notice that resources provisionned for the EF class that are not used are available for the other classes.

## 2.3 Network Model

The considered network is a DiffServ domain (all routers are DiffServ-compliant). Moreover, all routers in the domain are Label Switching Routers (LSR). Links interconnecting

routers are supposed to be FIFO and the transmission delay between two nodes has known lower and upper bounds:  $P_{min}$  and  $P_{max}$ .

Two DiffServ domain parameters are defined for the EF class:

- $D$ , the worst case end-to-end response time guarantee,
- $J$ , the worst case end-to-end delay jitter guarantee.

In this paper, we consider neither network failures nor packet losses.

## 2.4 Traffic Model

We focus on the flows belonging to the EF class. We consider a set  $\tau = \{\tau_1, \dots, \tau_n\}$  of  $n$  sporadic flows belonging to the EF class. Each flow  $\tau_i$  follows a sequence of nodes whose first node is the ingress node of the flow. In the following, we call line this sequence. Moreover, a sporadic flow  $\tau_i$  is defined by:

- $T_i$ , the minimum interarrival time (abusively called period) between two successive packets of  $\tau_i$ ;
- $C_i^h$ , the maximum processing time on node  $h$  of a packet of  $\tau_i$ ;
- $J_{in_i}^1$ , the maximum jitter of packets of  $\tau_i$  arriving in the DiffServ domain.

This characterization is well adapted to real-time flows (e.g. process control, voice and video, sensor and actuator). Any flow  $\tau_i \in \tau$  must meet the following constraints:

**Constraint 1.** *The end-to-end response time of any packet of  $\tau_i$  from the ingress node to any egress node must not exceed  $D$ .*

**Constraint 2.** *When leaving a DiffServ domain, the difference between end-to-end response times experienced by any two packets of  $\tau_i$  must not exceed  $J$ .*

For any packet  $g$  belonging to the EF class, we denote  $\tau(g)$  the index number of the EF flow which  $g$  belongs to. Moreover,  $B^h$  denotes the maximum processing time at node  $h$  of any packet of flow not belonging to the EF class.

## 3 Related Work

In this section, we first examine the existing approaches to obtain deterministic end-to-end response time guarantees in a multi-hop network. Then we see how these approaches can be applied to the Expedited Forwarding class of the DiffServ model. We then present works related to Diffserv and MPLS.

### 3.1 End-to-End Response Time

To determine the maximum end-to-end response time, several approaches can be used: a stochastic or a deterministic one. A *stochastic approach* consists in determining the mean behavior of the considered network, leading to mean, statistical or probabilistic end-to-end response times [21, 23]. A *deterministic approach* is based on a worst case analysis of the network behavior, leading to worst case end-to-end response times [9, 12].

In this paper, we are interested in the deterministic approach as we want to provide a deterministic guarantee of worst case end-to-end response times for any *ef* flow in the network. In this context, two different approaches can be used to determine the worst case end-to-end delay: the holistic approach and the trajectory approach.

- The *holistic approach* [22] considers the worst case scenario on each node visited by a flow, accounting for the maximum possible jitter introduced by the previous visited nodes. If no jitter control is done, the jitter will increase throughout the visited nodes. In this case, the minimum and maximum response times on a node  $h$  induce a maximum jitter on the next visited node  $h + 1$  that leads to a worst case response time and then a maximum jitter on the following node and so on. Otherwise, the jitter can be either cancelled or constrained.
  - the *Jitter Cancellation technique* consists in cancelling, on each node, the jitter of a flow before it is considered by the node scheduler [11]: a flow packet is held until its latest possible reception time. Hence a flow packet arrives at node  $h + 1$  with a jitter depending only on the jitter introduced by the previous node  $h$  and the link between them. As soon as this jitter is cancelled, this packet is seen by the scheduler of node  $h + 1$ . The worst case end-to-end response time is obtained by adding the worst case response time, without jitter (as cancelled) on every node;
  - the *Constrained Jitter technique* consists in checking that the jitter of a flow remains bounded by a maximum acceptable value before the flow is considered by the node scheduler. If not, the jitter is reduced to the maximum acceptable value by means of traffic shaping.

In conclusion, the holistic approach can be pessimistic as it considers worst case scenarios on every node possibly leading to impossible scenarios.

- The *trajectory approach* [16] consists in examining the scheduling produced by all the visited nodes of a flow. In this approach, only possible scenarios are examined. For instance, the fluid model (see [18] for GPS) is relevant to the trajectory approach. This approach produces the best results as no impossible scenario is considered but is somewhat more complex to use. This approach can also be used in conjunction with a jitter control (see [10] for EDF, and [18] for GPS). In this paper, we adopt the trajectory approach without jitter control in a DiffServ domain to determine the maximum end-to-end response time of an EF flow.

We can also distinguish two main traffic models: the sporadic model and the token bucket model. The sporadic model has been used in the holistic approach and in the trajectory approach, while the token bucket model has been used in the trajectory approach.

- The *sporadic model* is classically defined by three parameters: the processing time, the minimum interarrival time and the maximum release jitter, (see section 2.4). This model is natural and well adapted for real-time applications.
- The *token bucket* [9, 10, 18] is defined by two parameters:  $\sigma$ , the bucket size, and  $\rho$ , the token throughput. The token bucket can model a flow or all flows of a DiffServ class. In the first case, it requires to maintain per flow information on every visited node. This solution is not scalable. In the second case, the choice of good values for the token bucket parameters is complex when flows have different characteristics. With this model, arises the problem of fixing the good values of these parameters

for a given application. As shown in [10] and [20], the end-to-end response times strongly depend on the choice of the token bucket parameters. A bad choice can lead to bad response times. Furthermore, the token bucket parameters can be optimized for a given configuration, only valid at a given time. If the configuration evolves, the parameters of the token bucket should be recomputed on every node to remain optimal. This is not generally done.

In this paper, we adopt the trajectory approach with a sporadic traffic model.

### 3.2 Expedited Forwarding Class

The definition of the EF PHB as given in [15] can be used to predict qualitative end-to-end delay guarantees. Beyond this definition, end-to-end guarantees are crucial for delay and jitter sensitive applications. Those QoS guarantees must be provided without requiring per flow processing in the core routers. Otherwise the solution would not be scalable. [5] shows that the worst case delay jitter for the EF traffic can be large in case of large networks.

The use of the FIFO scheduling algorithm for priority traffic in a network based on traffic aggregation (e.g. all flows in the EF class share a single FIFO queue) has been discussed in [8]. Nevertheless, the found delay bound is valid only for reasonably small EF traffic utilization.

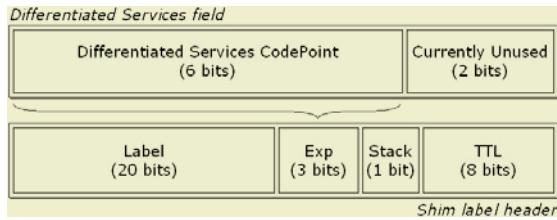
In [13], a hybrid admission control scheme has been proposed, based on traffic shaping at border routers, to provide QoS guarantees in a DiffServ environment. The decision of the admission control is based on measurements realized to estimate resource allocation, leading to a higher utilization efficiency. As we are interested in deterministic guarantees, the admission control accounts for the worst case response times and jitters.

### 3.3 MPLS and DiffServ

MPLS forwards a packet based on its label, while IP routing forwards a packet based on its IP header, and notably the destination address. Label switching technology reduces forwarding delays due to simpler processing. Labels are carried in a *shim label header*, inserted either between the layer-2 and layer-3 headers, or directly in the layer-2 header. Moreover, while current IP routing does not guarantee that two packets of the same flow follow the same path between two endpoints, MPLS does. Label switching technique with explicit routing capabilities allows traffic engineering by introducing the concept of a *traffic trunk* [3], which is a set of flows aggregated by their service class and then placed on a *Label Switched Path* (LSP).

At each LSR along an LSP, a packet is forwarded to the next hop based on its label. A solution for supporting the DiffServ Behavior Aggregates over a MPLS network is specified in [17]. Indeed, the 3-bit *Experimental* (Exp) field can encode, into the label header, the *Differentiated Services Code Point* (DSCP) which is used in each DiffServ-compliant router to select a predefined PHB. Since the DSCP requires six bits, two methods are proposed:

- if fewer than eight PHB are supported in the network, the Exp field is sufficient and its value can be mapped to the PHB. A LSP established in this way is called E-LSP (see Figure 2);



**Fig. 2.** Encoding of the DSCP into the Exp field

- if more than eight PHB are supported in the network, the DSCP has to be encoded in the label. A LSP, called L-LSP when supporting such an approach, must be set up for each PHB class. In this case, the Exp field can provide the drop precedence of packets in the *Assured Forwarding* service class.

## 4 Proposed Solution

For the problem defined in section 2, we propose a solution based on a combination of DiffServ and MPLS, where the EF class has the highest priority among the other classes. As at most eight classes are provided, the DSCP is encoded into the experimental field of the label. This solution is very simple for the following reasons:

- for a given destination, we need only one label. Hence the size of the label table is reduced;
- the QoS requirements are directly encoded into the label. Hence, the QoS routing is completely achieved at layer two, once the route has been established;
- each LSR uses a FIFO scheduling for the EF flows. FIFO scheduling is native in a DiffServ domain;
- the solution we propose requires neither to maintain information per EF flow nor to use traffic shaper in the core of the domain, leading to less complexity. The obtained solution can thus easily be implemented in a DiffServ domain.

The egress routers are in charge of ensuring that for any flow, the end-to-end jitter remains bounded by  $J$ . If it is not the case, the flow is reshaped to be compliant. This requires to synchronize clocks of the ingress and egress routers to control the delay jitter at the end of the domain.

The upper bound on the end-to-end response time we give enables to derive a simple admission control. Indeed, the admission of a new flow  $\tau_i$  in the EF class only requires a computation involving each node visited by  $\tau_i$ . The conditions that have to be checked concern (i) the workload, (ii) the maximum sojourn time and (iii) the end-to-end response time of the EF flows visiting at least one node visited by  $\tau_i$ . The solution with its associated admission control (see section 6) enables to ensure deterministic end-to-end response times for the EF flows in a DiffServ domain.

We now describe how a new EF flow is processed with our solution. When a new flow enters a DiffServ domain, the DSCP given in the DiffServ IP header determines the label appended to the packets of this flow. If this flow belongs to the EF class, the admission

control is called. The implementation of the admission control is not presented in this paper. It can be centralized with a bandwidth broker. It can also be distributed on nodes visited by (i) the new EF flow or (ii) a crossing EF flow. If accepted, the DiffServ domain guarantees the domain deadline  $D$  and jitter  $J$ , as defined in section 2.3. MPLS is then used for packet forwarding.

## 5 Worst Case End-to-End Real-Time Analysis

In this worst case analysis, we assume that time is discrete. [4] shows that results obtained with a discrete scheduling are as general as those obtained with a continuous scheduling when all flow parameters are multiples of the node clock tick. In such conditions, any set of flows is feasible with a discrete scheduling if and only if it is feasible with a continuous scheduling.

In this section, we focus on the end-to-end response time of any packet  $m$  of any EF flow  $\tau_i \in \{\tau_1, \dots, \tau_n\}$  following a *line*  $\mathcal{L}$ , where  $\mathcal{L}$  is a sequence of  $q$  nodes numbered from 1 to  $q$ . We first introduce the notations and definitions used throughout the paper and define the constituent parts of the end-to-end response time of any EF flow. Then, we show how to compute upper bounds on the end-to-end response time of any flow belonging to the EF class, based on a worst case analysis of FIFO scheduling.

### 5.1 Notations and Definitions

In the rest of this paper, we use the following notations and definitions:

$e$	element of the network such as a node, a line or the DiffServ domain;
$a_m^e$	the arrival time of packet $m$ in element $e$ ;
$d_m^e$	the departure time of packet $m$ from element $e$ ;
$R_m^e$	the response time of packet $m$ in element $e$ ;
$R_m^{e_1, e_2}$	the response time of packet $m$ between its arrival time in $e_1$ and its departure time on $e_2$ ;
$S_m^e$	the time taken by packet $m$ to arrive in element $e$ ;
$S_m^{e_1, e_2}$	the time taken by packet $m$ to go from $e_1$ to $e_2$ ;
$J_{in_i}^e$	the worst case jitter of flow $\tau_i$ when entering element $e$ ;
$J_{out_i}^e$	the worst case jitter of flow $\tau_i$ when leaving element $e$ ;
$P_m^{h, h+1}$	the network delay experienced by packet $m$ between nodes $h$ and $h + 1$ .

Moreover, we denote  $R_{min_i}^e$  (resp.  $R_{max_i}^e$ ) the minimum (resp. the maximum) response time experienced by packets of flow  $\tau_i$  in element  $e$ . Thus, we have:  $\forall m$  of  $\tau_i$ ,  $R_{min_i}^e \leq R_m^e \leq R_{max_i}^e$ . In the same way, we denote  $S_{min_i}^e$  (resp.  $S_{max_i}^e$ ) the minimum (resp. the maximum) delay experienced by packets of flow  $\tau_i$  from their arrival times in the domain to reach element  $e$ . Thus, we have:  $\forall m$  of  $\tau_i$ ,  $S_{min_i}^e \leq S_m^e \leq S_{max_i}^e$ . Finally, as assumed in section 2.3, we have:  $\forall m$  of  $\tau_i$ ,  $P_{min} \leq P_m^{h, h+1} \leq P_{max}$ .

**Definition 1.** An idle time  $t$  is a time such that all packets arrived before  $t$  have been processed at time  $t$ .

**Definition 2.** A busy period is defined by an interval  $[t, t')$  such that  $t$  and  $t'$  are both idle times and there is no idle time  $\in (t, t')$ .

**Definition 3.** For any node  $h$ , the processor utilization factor for the EF class is denoted  $U_{EF}^h$ . It is the fraction of processor time spent by node  $h$  in the processing of EF packets. It is equal to  $\sum_{i=1}^n (C_i^h / T_i)$ .

## 5.2 Constituent Parts of the End-to-End Response Time

Let us consider any EF flow  $\tau_i$ ,  $i \in [1, n]$ , following a line  $\mathcal{L}$ , where  $\mathcal{L}$  consists of  $q$  nodes numbered from 1 to  $q$ . The end-to-end response time of any packet  $m$  of  $\tau_i$  depends on its sojourn times on each visited node and network delays. Thus, we have:

$$R_m^{\mathcal{L}} = \sum_{h=1}^q R_m^h + \sum_{h=1}^{q-1} P_m^{h, h+1}.$$

As detailed in section 2.2, packet scheduling is non-preemptive. Hence, whatever the scheduling algorithm in the EF class and despite the highest priority of this class, a packet from another class (i.e. best-effort or AF class) can interfere with EF flows processing due to non-preemption. Indeed, if any EF packet  $m$  enters node  $h \in [1, q]$  while a packet  $m'$  not belonging to the EF class is being processed on this node,  $m$  has to wait until  $m'$  completion.

Generally, the non-preemptive effect may lead to consider EF packets with a priority higher than  $m$  arrived after  $m$ , but during  $m'$  execution. As we consider in this paper that the scheduling of the EF class is FIFO, no EF packet has priority over  $m$  if arrived after  $m$ . Hence, we get the following property.

**Property 1.** The maximum delay due to packets not belonging to the EF class incurred by any packet of the EF flow  $\tau_i$  is bounded by:  $\sum_{h=1}^q (B^h - 1)$ , where  $B^h$  denotes the maximum processing time at node  $h$  of any packet of flow not belonging to the EF class. By convention,  $B^h - 1 = 0$  if there is no such packets.

*Proof:* Let us show that if  $m$  is delayed on any node  $h$  of line  $\mathcal{L}$  by a packet  $m'$  not belonging to the EF class, then in the worst case the packet  $m$  arrives on node  $h$  just after the execution of  $m'$  begins. Indeed, by contradiction, if  $m$  arrives on the node before or at the same time as  $m'$ , then as  $m$  belongs to the EF class and this class has the highest priority,  $m$  will be processed before  $m'$ . Hence a contradiction. Thus, in the worst case,  $m$  arrives on node  $h$  one time unit after the execution of  $m'$  begins, because we assume a discrete time. The maximum processing time for  $m'$  on  $h$  is  $B^h$ . By convention, if all packets visiting node  $h$  belong to the EF class, we note  $B^h - 1 = 0$ . Hence, in any case, the maximum delay incurred by  $m$  on  $h$  directly due to a packet not belonging to the EF class is equal to  $B^h - 1$ . As we make no particular assumption concerning the classes other than the EF class, in the worst case packet  $m$  is delayed for  $B^h - 1$  on any visited node  $h$ . ■

Obviously, the execution of packet  $m$  on any node  $h \in [1, q]$  can also be delayed by packets belonging to the EF class and following a line  $\mathcal{L}'$  with  $\mathcal{L}' \cap \mathcal{L} \neq \emptyset$ . If we denote

$X_{EF}$  this maximum resulting delay, then the end-to-end response time of packet  $m$  of flow  $\tau_i$  is bounded by:

$$R_m^{\mathcal{L}} \leq X_{EF} + \sum_{h=1}^q (B^h - 1) + (q - 1) \cdot P_{max}.$$

In the following, we show how to compute the end-to-end response time of any packet  $m$  of flow  $\tau_i$ , based on a worst case analysis. This analysis enables to evaluate  $X_{EF}$ , the delay due to other EF packets.

### 5.3 Determination of the End-to-End Response Time

In this section, we show how to compute the worst case end-to-end delay for an EF packet  $m$  as it visits the DiffServ domain. The strategy is to move backwards through the sequence of nodes  $m$  visits, each time identifying preceding packets and busy periods that ultimately affect the delay of  $m$ . We proceed in that way till finally at node 1, the very first packet  $f(1)$  to affect  $m$ 's delay is identified. Thereafter, the worst case delay is estimated from the time that  $f(1)$  enters the domain to the time that  $m$  exits the domain. This, when subtracted by the difference between the arrival times of  $m$  and  $f(1)$  to the domain, yields the worst case delay for  $m$ .

#### 5.3.1 All the EF Flows Follow the Same Line in the DiffServ Domain

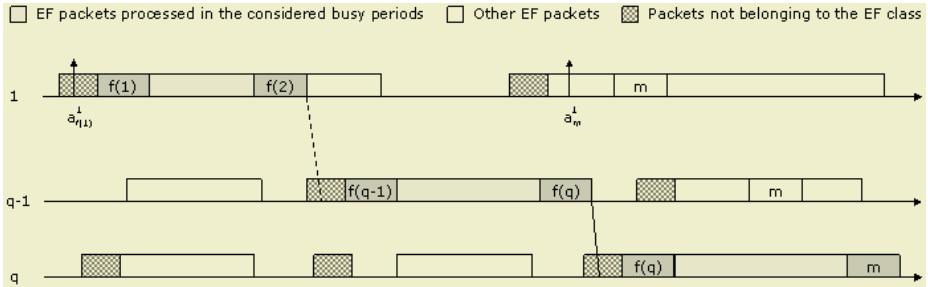
We assume in this subsection that all the EF flows follow the same line  $\mathcal{L}$  consisting of  $q$  nodes numbered from 1 to  $q$ . This assumption allows to clearly illustrate the strategy adopted to determine the worst case end-to-end response time of any EF flow. In subsection 5.3, we show how to generalize this result.

##### Decomposition of the End-to-End Response Time of $m$

Let us consider any packet  $m$  belonging to any EF flow  $\tau_i \in \tau$ . To compute the end-to-end response time of packet  $m$ , we identify the busy periods that affect the delay of  $m$ . For this, we consider the busy period  $bp^q$  in which  $m$  is processed on node  $q$  and we define  $f(q)$  as the first EF packet processed in  $bp^q$ .

The packet  $f(q)$  has been processed in the busy period  $bp^{q-1}$  on node  $q-1$ . We then define  $f(q-1)$  as the first EF packet processed in  $bp^{q-1}$ . And so on until the busy period of node 1 in which the packet  $f(1)$  is processed (see figure 3). In the worst case, on any node  $h \neq 1$ , packet  $f(h)$  arrives on node  $h$  one time unit after the beginning of  $bp^h$ .

For the sake of simplicity, on a node  $h$ , we number consecutively the EF packets entering the domain between the arrival times  $a_{f(1)}^1$  and  $a_m^1$  of  $f(1)$  and  $m$  respectively. Hence, on node  $h$ , we denote  $m'-1$  (resp.  $m'+1$ ) the packet preceding (resp. succeeding)  $m'$ . By adding parts of the considered busy periods, we can now express the latest departure time of packet  $m$  from node  $q$ , that is:



**Fig. 3.** Response time of packet  $m$

$$\begin{aligned}
 & a_{f(1)}^1 + (B^1 - 1) + \text{the processing time on node 1 of packets } f(1) \text{ to } f(2) + P_{f(2)}^{1,2} \\
 & + (B^2 - 1) + \text{the processing time on node 2 of packets } f(2) \text{ to } f(3) + P_{f(3)}^{2,3} \\
 & \dots \\
 & + (B^q - 1) + \text{the processing time on node } q \text{ of packets } f(q) \text{ to } m.
 \end{aligned}$$

By convention,  $f(q+1) = m$ . The end-to-end response time of packet  $m$  is then bounded by:

$$a_{f(1)}^1 - a_m^1 + \sum_{h=1}^q \left( \sum_{g=f(h)}^{f(h+1)} C_{\tau(g)}^h \right) + \sum_{h=1}^q (B^h - 1) + (q - 1) \cdot P_{max}.$$

#### Evaluation of the Maximum Delay due to EF Packets

We now consider the term  $X_{EF} = a_{f(1)}^1 - a_m^1 + \sum_{h=1}^q (\sum_{g=f(h)}^{f(h+1)} C_{\tau(g)}^h)$ , that is the maximum delay due to EF packets and incurred by  $m$ . We denote *slow* the slowest node of line  $\mathcal{L}$ . That is for any flow  $\tau_j$  visiting *slow*, for any node  $h$  visited by  $\tau_j$ , we have  $C_j^h \leq C_j^{slow}$ . We then distinguish the nodes visited by the EF flows before *slow* and those visited after. Thus, we have  $X_{EF}$  bounded by  $a_{f(1)}^1 - a_m^1$ , plus:

$$\begin{aligned}
 & \underbrace{\sum_{h=1}^{slow-1} \left( \sum_{g=f(h)}^{f(h+1)-1} C_{\tau(g)}^h + C_{\tau(f(h+1))}^h \right)}_{\text{nodes visited before slow}} + \underbrace{\sum_{g=f(slow)}^{f(slow+1)} C_{\tau(g)}^{slow}}_{\text{node slow}} \\
 & + \underbrace{\sum_{h=slow+1}^q \left( \sum_{g=f(h)+1}^{f(h+1)} C_{\tau(g)}^h + C_{\tau(f(h))}^h \right)}_{\text{nodes visited after slow}}.
 \end{aligned}$$

For any node  $h \in [1, q]$ , for any EF packet  $m'$  visiting  $h$ , the processing time of  $m'$  on node  $h$  is less than  $C_{\tau(m')}^{slow}$ . Then, as packets are numbered consecutively from  $f(1)$

to  $f(q+1) = m$ , we get inequation (1). By considering that on any node  $h$ , the processing time of an EF packet is less than or equal to  $C_{max}^h = \max_{j=1..n}(C_j^h)$ , we get inequation (2).

$$\begin{aligned} & \sum_{h=1}^{slow-1} \left( \sum_{g=f(h)}^{f(h+1)-1} C_{\tau(g)}^h \right) + \sum_{g=f(slow)}^{f(slow+1)} C_{\tau(g)}^{slow} + \sum_{h=slow+1}^q \left( \sum_{g=f(h)+1}^{f(h+1)} C_{\tau(g)}^h \right) \\ & \leq \sum_{g=f(1)}^m C_{\tau(g)}^{slow} \end{aligned} \quad (1)$$

$$\sum_{h=1}^{slow-1} C_{\tau(f(h+1))}^h + \sum_{h=slow+1}^q C_{\tau(f(h))}^h \leq \sum_{\substack{h=1 \\ h \neq slow}}^q C_{max}^h \quad (2)$$

By (1) and (2), we have:  $X_{EF} \leq a_{f(1)}^1 - a_m^1 + \sum_{g=f(1)}^m C_{\tau(g)}^{slow} + \sum_{\substack{h=1 \\ h \neq slow}}^q C_{max}^h$ . The term  $\sum_{g=f(1)}^m C_{\tau(g)}^{slow}$  is bounded by the maximum workload generated by the EF flows in the interval  $[a_{f(1)}^1, a_m^1]$ . By definition, this quantity is equal to:  $\sum_{j=1}^n (1 + \lfloor (a_m^1 - a_{f(1)}^1 + J_{in_j}^1)/T_j \rfloor) \cdot C_j^{slow}$  (see [11]). As for any  $x$ ,  $\lfloor x \rfloor \leq x$ , we can write:

$$\begin{aligned} \sum_{g=f(1)}^m C_{\tau(g)}^{slow} & \leq \sum_{j=1}^n \left( 1 + \frac{a_m^1 - a_{f(1)}^1 + J_{in_j}^1}{T_j} \right) \cdot C_j^{slow} = \\ & \sum_{j=1}^n \left( 1 + \frac{J_{in_j}^1}{T_j} \right) \cdot C_j^{slow} + (a_m^1 - a_{f(1)}^1) \cdot U_{EF}^{slow}, \end{aligned}$$

where  $U_{EF}^{slow}$  denotes the processor utilization factor for the EF class on node  $slow$  and must be  $\leq 1$ .

Consequently,  $X_{EF} \leq \sum_{j=1}^n (1 + J_{in_j}^1/T_j) \cdot C_j^{slow} + (a_m^1 - a_{f(1)}^1) \cdot (U_{EF}^{slow} - 1) + \sum_{\substack{h=1 \\ h \neq slow}}^q C_{max}^h$ , that is bounded by:  $\sum_{j=1}^n (1 + J_{in_j}^1/T_j) \cdot C_j^{slow} + \sum_{\substack{h=1 \\ h \neq slow}}^q C_{max}^h$ .

### Worst Case End-to-End Response Time of an EF Packet

We have shown in subsection 5.2 that  $R_m^{\mathcal{L}}$ , the end-to-end response time of any packet  $m$  of flow  $\tau_i$ ,  $i \in [1, n]$ , is bounded by:  $X_{EF} + \sum_{h=1}^q (B^h - 1) + (q - 1) \cdot P_{max}$ . Hence, considering the above defined bound for  $X_{EF}$ , we finally get:

**Property 2.** *When all the EF flows follow the same line  $\mathcal{L}$  consisting of  $q$  nodes numbered from 1 to  $q$ , then the worst case end-to-end response time of any EF flow  $\tau_i$  meets:*

$$R_{max,i}^{\mathcal{L}} \leq \sum_{j=1}^n \left( 1 + \frac{J_{in_j}^1}{T_j} \right) \cdot C_j^{slow} + \sum_{\substack{h=1 \\ h \neq slow}}^q C_{max}^h + \sum_{h=1}^q (B^h - 1) + (q - 1) \cdot P_{max}.$$

### 5.3.2 Extension to Several Lines

In the previous subsection, we assumed that all the EF flows followed the same line (i.e. the same sequence of nodes). In this subsection, we extend the obtained results to the general case, that is EF flows can follow different lines. Let us consider any EF flow  $\tau_i \in \tau$  following line  $\mathcal{L}$ , that consists of  $q$  nodes numbered from 1 to  $q$ . To determine the worst case end-to-end response time of any packet  $m$  of  $\tau_i$ , we adopt the same strategy as the one used in subsection 5.3, that consists in identifying the busy periods that affect the delay of  $m$  on the nodes visited by  $\tau_i$ . For the sake of simplicity, we first consider that any EF flow  $\tau_j$  following line  $\mathcal{L}'$  with  $\mathcal{L}' \neq \mathcal{L}$  and  $\mathcal{L}' \cap \mathcal{L} \neq \emptyset$  never visits a node of line  $\mathcal{L}$  after having left line  $\mathcal{L}$  (cf. assumption 1). In subsection 5.3, we show how to remove this assumption.

**Assumption 1.** *For any EF flow  $\tau_j$  following line  $\mathcal{L}'$  with  $\mathcal{L}' \neq \mathcal{L}$  and  $\mathcal{L}' \cap \mathcal{L} \neq \emptyset$ , if there exists a node  $h \in \mathcal{L}' \cap \mathcal{L}$  such that  $\tau_j$  visits  $h' \neq h + 1$  immediately after  $h$ , then  $\tau_j$  never visits a node  $h'' \in \mathcal{L}$  after.*

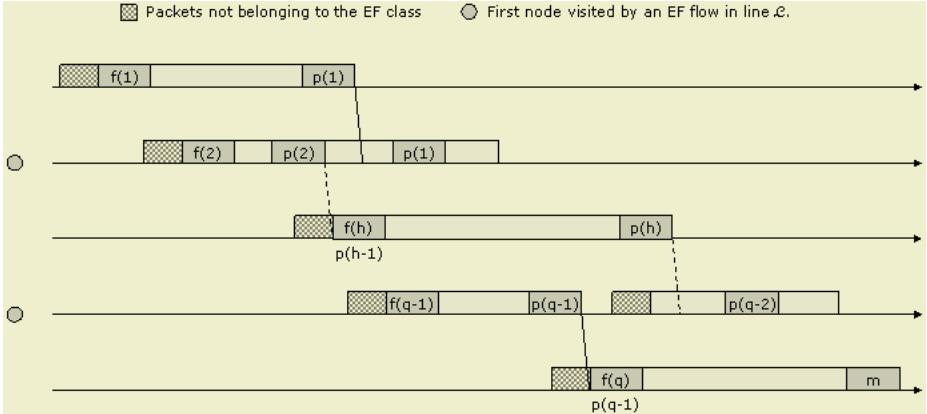
#### Decomposition of the End-to-End Response Time of $m$

Let us consider any packet  $m$  belonging to any EF flow  $\tau_i \in \tau$  and following line  $\mathcal{L}$ , where  $\mathcal{L}$  consists of  $q$  nodes numbered from 1 to  $q$ . Identifying the busy periods that affect the delay of  $m$  on the nodes visited by  $\tau_i$  is somewhat more complex than in subsection 5.3, where all the EF flows followed the same line. Indeed, we still focus on the busy period  $bp^q$  in which  $m$  is processed on node  $q$  and we define  $f(q)$  as the first EF packet processed in  $bp^q$ . But contrary to the single line case,  $f(q)$  does not necessarily come from node  $q - 1$  since the EF flow that  $f(q)$  belongs to may follow a line different from  $\mathcal{L}$ .

Hence, to move backwards through the sequence of nodes  $m$  traverses, each time identifying preceding packets and busy periods that ultimately affect the delay of  $m$ , we have to consider an additional packet on node  $q$ , that is  $p(q - 1)$ : the first EF packet processed between  $f(q)$  and  $m$  on node  $q$  and coming from node  $q - 1$ . We denote  $bp^{q-1}$  the busy period in which  $p(q - 1)$  has been processed on node  $q - 1$  and  $f(q - 1)$  the first EF packet processed in  $bp^{q-1}$ . We then define  $p(q - 2)$  as the first EF packet processed between  $f(q - 1)$  and  $m$  on node  $q - 1$  and coming from node  $q - 2$ . And so on until the busy period of node 1 in which the packet  $f(1)$  is processed. We have thus determined the busy periods on nodes visited by  $\tau_i$  that can be used to compute the end-to-end response time of packet  $m$  (cf. figure 4).

The latest departure time of packet  $m$  from node  $q$  is then equal to  $a_{f(1)}^1$ , plus:

$$\begin{aligned}
 & (B^1 - 1) + \text{the processing time on node 1 of packets } f(1) \text{ to } p(1) + P_{p(1)}^{1,2} \\
 & + (B^2 - 1) + \text{the processing time on node 2 of packets } f(2) \\
 & \quad \text{to } p(2) + P_{p(2)}^{2,3} - (a_{p(1)}^2 - a_{f(2)}^2) \dots \\
 & + (B^q - 1) + \text{the processing time on node } q \text{ of packets } f(q) \\
 & \quad \text{to } m - (a_{p(q-1)}^q - a_{f(q)}^q).
 \end{aligned}$$

**Fig. 4.** Response time of packet  $m$ 

Hence, the end-to-end response time of packet  $m$  is bounded by:

$$a_{f(1)}^1 - a_m^1 + \sum_{h=1}^q \left( \sum_{g=f(h)}^{p(h)} C_{\tau(g)}^h \right) + \sum_{h=1}^q (B^h - 1) + (q-1) \cdot P_{max} - \sum_{h=2}^q (a_{p(h-1)}^h - a_{f(h)}^h).$$

Let  $first_j$  denote the first node visited by the EF flow  $\tau_j$  in line  $\mathcal{L}$ . We can notice that on any node  $h$  of line  $\mathcal{L}$ , if there exists no EF flow  $\tau_j$  such that  $h = first_j$ , then  $p(h-1) = f(h)$  and so  $a_{p(h-1)}^h - a_{f(h)}^h = 0$ . In other words, if  $p(h-1) \neq f(h)$ , then there exists an EF flow  $\tau_j$  such that  $h = first_j$ . In such a case, by definition of  $p(h)$ , all the packets in  $[f(h), p(h-1)]$  cross line  $\mathcal{L}$  for the first time at node  $h$ . We can then act on their arrival times. Postponing the arrivals of these messages in the busy period where  $p(h-1)$  is processed, would increase the departure time of  $m$  from node  $q$ . Hence, in the worst case,  $p(h-1) \in bp^h$  and  $a_{p(h-1)}^h = a_{f(h)}^h$ .

Thus, in the worst case, we obtain a decomposition similar to the one presented in subsection 5.3. By numbering consecutively on any node  $h$  the EF packets processed after  $f(h)$  and before  $p(h)$  (with  $p(q) = m$ ), we get an upper bound on the end-to-end response time of packet  $m$ , that is:

$$a_{f(1)}^1 - a_m^1 + \sum_{h=1}^q \left( \sum_{g=f(h)}^{p(h)} C_{\tau(g)}^h \right) + \sum_{h=1}^q (B^h - 1) + (q-1) \cdot P_{max}.$$

#### Evaluation of the Maximum Delay due to EF Packets

We now consider the term  $X_{EF} = a_{f(1)}^1 - a_m^1 + \sum_{h=1}^q \left( \sum_{g=f(h)}^{p(h)} C_{\tau(g)}^h \right)$ , that is the maximum delay due to EF packets and incurred by  $m$ . We denote  $slow$  the slowest node among the  $q$  nodes visited by  $m$ . That is for any flow  $\tau_j$  visiting  $slow$ , for any node  $h$  visited by  $\tau_j$  we have  $C_j^h \leq C_j^{slow}$ .

By definition, for any node  $h \in [1, slow]$ ,  $p(h)$  is the first packet belonging to the EF class, processed in  $bp^{h+1}$  and coming from node  $h$ . Moreover,  $p(h)$  is the last packet considered in  $bp^h$ . Hence, if we count packets processed in  $bp^h$  and  $bp^{h+1}$ , only  $p(h)$  is counted twice. In the same way, for any node  $h \in (slow, q]$ ,  $p(h-1)$  is the first packet belonging to the EF class, processed in  $bp^h$  and coming from node  $h-1$ . Moreover,  $p(h-1)$  is the last EF packet considered in  $bp^{h-1}$ . Thus,  $p(h-1)$  is the only packet counted twice when counting packets processed in  $bp^{h-1}$  and  $bp^h$ .

In addition, for any node  $h \in [1, q]$ , for any EF packet  $g$  visiting  $h$ , the processing time of  $g$  on node  $h$  is less than  $C_{\tau(g)}^{slow_{\tau(g)}}$ , where  $slow_j$  is the slowest node visited by  $\tau_j$  on line  $\mathcal{L}$ . Hence, if on any node  $h \in [1, slow)$  (resp.  $h \in (slow, q]$ ), we bound  $p(h)$  (resp.  $p(h-1)$ ) by  $C_{max}^h = \max_{j \in \tau}(C_j^h)$ , then we get:

$$X_{EF} \leq a_{f(1)}^1 - a_m^1 + \sum_{g=f(1)}^m C_{\tau(g)}^{slow_{\tau(g)}} + \sum_{\substack{h=1 \\ h \neq slow}}^q C_{max}^h.$$

We now evaluate the quantity  $\sum_{g=f(1)}^m C_{\tau(g)}^{slow_{\tau(g)}}$ . This quantity is bounded by the maximum workload generated by the EF flows visiting at least one node of line  $\mathcal{L}$ .

On node  $h$ , a packet of any EF flow  $\tau_j$  visiting at least one node visited by  $\tau_i$  can delay the execution of  $m$  if it arrives on node  $first_j$ ,  $first_j$  denoting the first node common to  $\tau_j$  and  $\tau_i$ , at the earliest at time  $a_{f(first_j)}^{first_j}$  and at the latest at time  $a_m^{first_j}$ . Indeed, if a packet of  $\tau_j$  arrives on node  $first_j$  after  $a_m^{first_j}$ , it will be processed on this node after  $m$  due to the FIFO-based scheduling of the EF class. So, if the next node visited by  $m$  is also the next node visited by the packet of  $\tau_j$ , this packet will arrive after  $m$  on this node, and so on. Therefore, it will not delay packet  $m$ .

Thus, the packets of  $\tau_j$  that can delay the execution of  $m$  are those arrived on node  $first_j$  in the interval  $[a_{f(first_j)}^{first_j} - S_{max_j}^{first_j} - J_{in_j}^1, a_m^{first_j} - S_{min_j}^{first_j}]$ , where  $S_{max_j}^{first_j}$  (respectively  $S_{min_j}^{first_j}$ ) denotes the maximum time (respectively the minimum time) taken by a packet of flow  $\tau_j$  to arrive on node  $first_j$ . The maximum number of packets of  $\tau_j$  that can delay  $m$  is then equal to:

$$1 + \left\lfloor \frac{a_m^{first_j} - S_{min_j}^{first_j} - (a_{f(first_j)}^{first_j} - S_{max_j}^{first_j} - J_{in_j}^1)}{T_j} \right\rfloor.$$

By definition,  $S_{max_j}^{first_j} - S_{min_j}^{first_j} + J_{in_j}^1 = J_{in_j}^{first_j}$ , where  $J_{in_j}^{first_j}$  denotes the delay jitter experienced by  $\tau_j$  between its source node and node  $first_j$ . Applying this property to all the EF flows visiting at least one node visited by  $\tau_i$ , we get:

$$\sum_{g=f(1)}^m C_{\tau(g)}^{slow_{\tau(g)}} \leq \sum_{h=1}^q \left( \sum_{\substack{j=1 \\ first_j=h}}^n \left( 1 + \left\lfloor \frac{a_m^h - a_{f(h)}^h + J_{in_j}^h}{T_j} \right\rfloor \right) \cdot C_j^{slow_j} \right).$$

As  $a_m^h - a_{f(h)}^h \leq a_m^h - a_m^1 + a_m^1 - a_{f(1)}^1$ , we get  $a_{f(1)}^1 - a_m^1 + \sum_{g=f(1)}^m C_{\tau(g)}^{slow_{\tau(g)}}$  bounded by:

$$\begin{aligned}
& a_{f(1)}^1 - a_m^1 + \sum_{h=1}^q \left( \sum_{\substack{j=1 \\ first_j=h}}^n \left( 1 + \frac{a_m^h - a_m^1 + a_m^1 - a_{f(1)}^1 + Jin_j^h}{T_j} \right) \cdot C_j^{slow_j} \right) \\
& \leq \sum_{h=1}^q \left( \sum_{\substack{j=1 \\ first_j=h}}^n \left( 1 + \frac{S_m^h + Jin_j^h}{T_j} \right) \cdot C_j^{slow_j} \right) \\
& + \left( a_m^1 - a_{f(1)}^1 \right) \cdot \left( \sum_{h=1}^q \left( \sum_{\substack{j=1 \\ first_j=h}}^n \frac{C_j^{slow_j}}{T_j} \right) - 1 \right).
\end{aligned}$$

Hence, if  $\sum_{h=1}^q \left( \sum_{\substack{j=1 \\ first_j=h}}^n (C_j^{slow_j} / T_j) \right) \leq 1$ , the maximum delay due to EF packets and incurred by  $m$  is bounded by:

$$X_{EF} \leq \sum_{h=1}^q \left( \sum_{\substack{j=1 \\ first_j=h}}^n \left( 1 + \frac{S_m^h + Jin_j^h}{T_j} \right) \cdot C_j^{slow_j} \right) + \sum_{\substack{h=1 \\ h \neq slow}}^q C_{max}^h.$$

#### Worst Case End-to-End Response Time of an EF Packet

In subsection 5.2, we have defined  $R_m^{\mathcal{L}}$  as the end-to-end response time of any packet  $m$  of flow  $\tau_i$ ,  $i \in [1, n]$ , and shown that  $R_m^{\mathcal{L}} \leq X_{EF} + \sum_{h=1}^q (B^h - 1) + (q - 1) \cdot P_{max}$ . Hence, considering the above defined bound for  $X_{EF}$ , we finally get:

**Property 3.** *If the condition  $\sum_{h=1}^q \left( \sum_{\substack{j=1 \\ first_j=h}}^n (C_j^{slow_j} / T_j) \right) \leq 1$  is met, then the worst case end-to-end response time of any packet of EF flow  $\tau_i$  is bounded by:*

$$\begin{aligned}
R_{max_i}^{\mathcal{L}} & \leq \sum_{h=1}^q \left( \sum_{\substack{j=1 \\ first_j=h}}^n \left( 1 + \frac{S_{max_i}^h + Jin_j^h}{T_j} \right) \cdot C_j^{slow_j} \right) + \sum_{\substack{h=1 \\ h \neq slow}}^q C_{max}^h \\
& + \sum_{h=1}^q (B^h - 1) + (q - 1) \cdot P_{max}.
\end{aligned}$$

where  $slow_j$  is the slowest node visited by  $\tau_j$  on line  $\mathcal{L}$  and  $first_j$  is the first node visited by  $\tau_j$  on line  $\mathcal{L}$ .

In section 6, we will see how to bound  $S_{max_i}^h$ , the maximum delay experienced by packets of flow  $\tau_i$  from their arrival times in the domain to reach node  $h$  and  $Jin_j^h$ , the delay jitter experienced by flow  $\tau_j$  between its source node and node  $h$ .

We can notice that this bound on the worst case end-to-end response time is more accurate than the sum of the maximum sojourn times on the visited nodes, plus the sum of the maximum network delays. This shows the interest of the trajectory approach comparatively to the holistic one.

#### **Remark: Particular Case of the Same Line**

In the particular case of a single line, we notice that:

- the condition given in Property 3 becomes  $\sum_{j=1}^n C_j^{slow}/T_j \leq 1$ . This condition is the local workload condition (see Lemma 1 in section 6);
- the bound on the end-to-end response time given in Property 3 becomes:

$$R_{max_i^{\mathcal{L}}} \leq \sum_{j=1}^n \left(1 + \frac{J_{in_j^1}}{T_j}\right) \cdot C_j^{slow} + \sum_{\substack{h=1 \\ h \neq slow}}^q C_{max}^h + \sum_{h=1}^q (B^h - 1) + (q-1) \cdot P_{max}.$$

### 5.3.3 Generalization

Property 3 can be extended to the general case by decomposing an EF flow in as many independent flows as needed to meet assumption 1. To achieve that, the idea is to consider an EF flow crossing line  $\mathcal{L}$  after it left  $\mathcal{L}$  as a new EF flow. We proceed by iteration until meeting assumption 1. We then apply Property 3 considering all these flows.

## 5.4 End-to-End Jitter

We have assumed that any flow  $\tau_i$  belonging to the EF class enters the DiffServ domain with a delay jitter bounded by  $J_{in_i^1}$ . But within the domain, the flow  $\tau_i$  will experience an additional jitter. To meet the constraint 2 given in Section 2, the last node visited by  $\tau_i$  in the considered DiffServ domain has to hold every  $\tau_i$  packet until its end-to-end response time is in the interval  $[R_{max_i^{\mathcal{L}}} - J, R_{max_i^{\mathcal{L}}}]$ , where  $R_{max_i^{\mathcal{L}}}$  is the worst case end-to-end response time of flow  $\tau_i$  and  $J$  the worst case end-to-end delay jitter guarantee.

## 6 Admission Control

We will now use the bound on the worst case response time for the EF flows to derive an admission control for the EF class. When a new flow belonging to the EF class arrives in the DiffServ domain, the admission control is in charge of deciding whether this flow can be accepted. The admission control has two major goals: (i) the new flow must not experience an end-to-end response time greater than the domain deadline, and (ii) the acceptance of the new flow must not compromise guarantees given to the already accepted EF flows. Notice that the domain jitter is enforced by traffic reshaping at the egress nodes (see section 5.4).

Moreover, the design of an admission control for the EF class is a complex task in so far as the admission request of a new EF flow  $\tau_i$  must not lead to recompute the worst case end-to-end response time of any EF flow already accepted. Indeed the acceptance of  $\tau_i$  results in an increased sojourn time of  $\tau_j$ , an already accepted EF flow crossing  $\tau_i$ . This leads to an increased jitter of  $\tau_j$  on the first node where  $\tau_j$  crosses  $\tau_k$  an already accepted EF flow not crossing  $\tau_i$ . This in turn increases the sojourn time of  $\tau_k$ , and so on.

To avoid this cascading effect, we define  $d_{\mathcal{L}'}^h$  as the maximum sojourn time in node  $h$  guaranteed to any EF flow  $\tau_j$  following  $\mathcal{L}'$ . The decomposition of the domain deadline  $D$  in intermediate deadlines is a general problem (see for instance [19]) and is outside the scope of this paper. The admission control uses this guaranteed sojourn time on node  $h$  to maximize the impact of any EF flow  $\tau_j$  on  $\tau_i$ . As long as the guaranteed sojourn time

of  $\tau_j$  is not violated, the guaranteed sojourn times as well as the end-to-end response time of any EF flow  $\tau_k$  not crossing  $\tau_i$  are not modified.

To reach its goals, the admission control has to verify the four following conditions given in table 1 when a new EF flow  $\tau_i$  enters the DiffServ domain to follow a line  $\mathcal{L}$ . It is important to underline that only the worst case end-to-end response times experienced by EF flows visiting at least one node visited by  $\tau_i$  have to be checked, and not those of all the EF flows in the DiffServ domain.

**Table 1.** Conditions to be checked by the admission control for the acceptance of the new EF flow  $\tau_i$

Local workload	$\forall h \in \mathcal{L}, U_{EF}^h \leq 1$
Distributed workload	$\sum_{h=1}^q \left( \sum_{\substack{j=1 \\ first_j=h}}^n \frac{C_j^{slow_j}}{T_j} \right) \leq 1$
Sojourn time	<ul style="list-style-type: none"> <li><math>\forall h \in \mathcal{L}, Rmax_i^h \leq d_{\mathcal{L}}^h</math></li> <li><math>\forall</math> EF flow <math>\tau_j</math> following <math>\mathcal{L}'</math>, <math>\forall h \in \mathcal{L}' \cap \mathcal{L}, Rmax_j^h \leq d_{\mathcal{L}'}^h</math></li> </ul>
End-to-end response time	<ul style="list-style-type: none"> <li><math>Rmax_i^{\mathcal{L}} \leq D</math></li> <li><math>\forall</math> EF flow <math>\tau_j</math> following <math>\mathcal{L}'</math> such that <math>\mathcal{L}' \cap \mathcal{L} \neq \emptyset, Rmax_j^{\mathcal{L}'} \leq D</math></li> </ul>

We now explain how each condition is checked by the admission control.

#### The Local Workload Condition

When a new EF flow  $\tau_i$  arrives in the DiffServ domain, the first condition the admission control must verify concerns the workload generated by all the EF flows on each node visited by  $\tau_i$ . This condition is given in lemma 1.

**Lemma 1.** *A necessary condition for the feasibility of a set of EF flows on a node  $h$ , in the presence of flows of other classes, is  $U_{EF}^h \leq 1$ .*

#### The Distributed Workload Condition

This is the condition required by property 3 .

#### The Sojourn Time Condition

If the workload conditions are met, the admission control checks that on any node  $h$  visited by the new EF flow  $\tau_i$ , this flow and each already accepted EF flow  $\tau_j$  following

$\mathcal{L}'$ , with  $h \in \mathcal{L}' \cap \mathcal{L}$ , meet their maximum guaranteed sojourn time on this node, that is:  $R_{max_i^h} \leq d_{\mathcal{L}}^h$  and  $R_{max_j^h} \leq d_{\mathcal{L}'}^h$ .

The maximum sojourn time of  $m$ , a packet of the EF flow  $\tau_i$ , consists of two parts. The first one is due to the non-preemption. As shown in subsection 5.2, this delay is less than or equal to  $B^h - 1$ . The second one is due to other EF packets arrived on node  $h$  before packet  $m$  and processed in the same busy period. For any flow  $\tau_j$  visiting  $h$ , we can bound the number of packets of  $\tau_j$  that can delay  $m$ . This quantity is equal to:

$$1 + \left\lfloor \frac{a_m^{first_j} - a_{f'_j(h)}^{first_j} + Jin_j^{first_j}}{T_j} \right\rfloor,$$

where  $f'_j(h)$  is the first packet of  $\tau_j$  processed on node  $h$  in the same busy period as  $m$ . This evaluation is similar to the one given in section 5.3. Then, we get:

$$\begin{aligned} R_m^h &\leq \min_{\tau_j \text{ visits } h} \left( a_{f'_j(h)}^h \right) - a_m^h + (B^h - 1) \\ &+ \sum_{\tau_j \text{ visits } h} \left( 1 + \left\lfloor \frac{a_m^{first_j} - a_{f'_j(h)}^{first_j} + Jin_j^{first_j}}{T_j} \right\rfloor \right) \cdot C_j^h. \end{aligned}$$

Moreover, we can write that the last term of this bound is equal to:

$$\sum_{\tau_j \text{ visits } h} \left( 1 + \left\lfloor \frac{a_m^{first_j} - \min_j \left( a_{f'_j(h)}^h \right) + \min_j \left( a_{f'_j(h)}^h \right) - a_m^h + a_m^h - a_{f'_j(h)}^{first_j} + Jin_j^{first_j}}{T_j} \right\rfloor \right) \cdot C_j^h.$$

Therefore, the sojourn time of packet  $m$  on node  $h$  is less than or equal to:

$$\begin{aligned} (B^h - 1) &+ \sum_{\tau_j \text{ visits } h} \left( 1 + \frac{a_m^{first_j} - a_m^h + a_{f'_j(h)}^h - a_{f'_j(h)}^{first_j} + Jin_j^{first_j}}{T_j} \right) \cdot C_j^h \\ &+ \left( a_m^h - \min_{\tau_j \text{ visits } h} \left( a_{f'_j(h)}^h \right) \right) \cdot \left( \sum_{j=1}^n \frac{C_j^h}{T_j} - 1 \right). \end{aligned}$$

As shown in lemma 1,  $\sum_{j=1}^n (C_j^h / T_j) = U_{EF}^h \leq 1$ . Moreover,  $a_m^h - a_m^{first_j} = S_m^{first_j, h}$ , that is the time taken by packet  $m$  to go from  $first_j$  to  $h$ . In the same way,  $a_{f'_j(h)}^h - a_{f'_j(h)}^{first_j} = S_{f'_j(h)}^{first_j, h}$ , that is the time taken by packet  $f'_j(h)$  to go from  $first_j$  to  $h$ . Finally, we get the following lemma.

**Lemma 2.** *For any EF flow  $\tau_i$  following line  $\mathcal{L}$ , for any node  $h$  visited by  $\tau_i$ , the sojourn time on node  $h$  of any packet of  $\tau_i$  meets:*

$$R_m^h \leq (B^h - 1) + \sum_{\substack{\tau_j \text{ follows } \mathcal{L}' \\ h \in \mathcal{L}' \cap \mathcal{L}}} \left( 1 + \frac{S_m^{first_j,h} - S_m^{first_j,h} + Jin_j^{first_j}}{T_j} \right) \cdot C_j^h.$$

where  $f'_j(h)$  denotes the first packet of the EF flow  $\tau_j$  processed on node  $h$  in the same busy period as  $m$ . Moreover,  $S_m^{first_j,h}$  (resp.  $S_{f'_j(h)}^{first_j,h}$ ) denotes the time taken by  $m$  (resp.  $f'_j(h)$ ) to arrive on node  $h$ , from node  $first_j$ .

Hence, the admission control has to check that:

- the bound on the worst case sojourn time of the new EF flow  $\tau_i$  is less than or equal to  $d_{\mathcal{L}'}^h$ ;
- the bound on the worst case sojourn time of any EF flow  $\tau_j$  following  $\mathcal{L}'$ , with  $h \in \mathcal{L}' \cap \mathcal{L}$ , is less than or equal to  $d_{\mathcal{L}'}^h$ .

A simple way to bound  $S_{f'_j(h)}^{first_j,h} - S_m^{first_j,h}$  is to maximize  $S_{f'_j(h)}^{first_j,h}$  by  $\sum_{k=first_j}^h d_{\mathcal{L}'}^h + \sum_{k=first_j}^{h-1} P_{max}$  and to minimize  $S_m^{first_j,h}$  by  $\sum_{k=first_j}^h C_i^h + \sum_{k=first_j}^{h-1} P_{min}$ .

### The End-to-End Response Time Condition

To meet the constraint 1 defined in subsection 2.4, the admission control finally checks that the end-to-end response times of  $\tau_i$  and of each already accepted EF flow  $\tau_j$  following line  $\mathcal{L}'$  such that  $\mathcal{L}' \cap \mathcal{L} \neq \emptyset$  meet the domain deadline  $D$ :  $R_{max_i^{\mathcal{L}'}} \leq D$  and  $R_{max_j^{\mathcal{L}'}} \leq D$ .

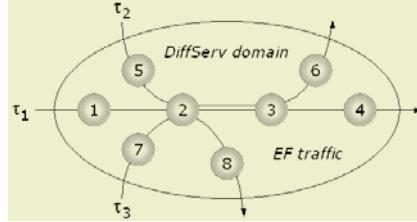
To compute the upper bound we have established in subsection 5.3 on the worst case end-to-end response time of any EF packet  $m$ , the admission control has to evaluate the terms  $Jin_j^h$  and  $Smax_i^h$ .

- $Jin_j^h$ , that is the delay jitter experienced by flow  $\tau_j$  between its source node and node  $h$ , is equal to:  $S_{max_j^h} - S_{min_j^h}$ . This quantity is bounded by:  $\sum_{k=1}^{h-1} (d_{\mathcal{L}'}^k - C_j^k) + (h-1) \cdot (P_{max} - P_{min})$ , where  $\mathcal{L}'$  is the line followed by  $\tau_j$ , composed of  $q'$  nodes numbered from 1 to  $q'$ .
- $Smax_i^h$ , that is the maximum delay experienced by packets of flow  $\tau_i$  from their arrival times in the domain to reach node  $h$ , is bounded by  $R_{max_i^{1,h-1}} + P_{max}$ , where  $R_{max_i^{1,h-1}}$  denotes the worst case response time experienced by packets of flow  $\tau_i$  between their arrival times on node 1 and their departure times on node  $h-1$ .  $R_{max_i^{1,h}}$  can be computed by iteration on the number of nodes of line  $\mathcal{L}$  in property 3. More precisely, we have:  $R_{max_i^{1,h}} = \mathcal{X}_i^h + \sum_{k=slow}^h C_{max}^k$ , where  $slow$  denotes the slowest node among nodes 1 to  $h$ , and:

$$\begin{cases} \mathcal{X}_i^1 = \sum_{\substack{j=1 \\ first_j=1}}^n \left( 1 + \frac{Jin_j^1}{T_j} \right) \cdot C_j^1 + (B^1 - 1), \\ \mathcal{X}_i^h = \mathcal{X}_i^{h-1} + \sum_{\substack{j=1 \\ first_j=h}}^n \left( 1 + \frac{S_{max_i^h} + Jin_j^h}{T_j} \right) \cdot C_j^{slow_j} + (B^h - 1) + P_{max}. \end{cases}$$

## 7 Example

In this section, we give an example of computation of the bound on the end-to-end response time established in subsection 5.3. We assume that three EF flows  $\tau_1$ ,  $\tau_2$  and  $\tau_3$  have already been accepted. As shown in figure 5,  $\tau_1$  visits nodes 1, 2, 3 and 4,  $\tau_2$  visits nodes 5, 2, 3 and 6, and  $\tau_3$  visits nodes 7, 2 and 8.



**Fig. 5.** EF traffic in the DiffServ domain

We want to admit a new flow  $\tau_4$  visiting line  $\mathcal{L}$  consisted of nodes 1, 2, 3 and 4. We assume that the DiffServ domain deadline is  $D = 100$ . All the flows  $\tau_i$ ,  $i = 1..4$ , have a period  $T_i = 10$  and a jitter  $J_{in}^i = 0$ . For any packet of these EF flows, the processing time on a core router is equal to 2, the processing time on an ingress/egress router is equal to 3.

Moreover, we assume that flows not belonging to the EF class exist in the DiffServ domain. The maximum processing time of a packet of these flows are equal to 2 on a core router and 3 on an ingress/egress router. The maximum guaranteed sojourn time for any EF packet is equal to 10 on any ingress node, 25 on any node in the core network and 40 on any egress node. Finally, we have  $P_{max} = P_{min} = 1$ .

1. The admission control first checks that on any node  $h = 1..4$ ,  $U_{EF}^h \leq 1$ . As  $U_{EF}^1 = U_{EF}^3 = U_{EF}^4 = 0.6$  and  $U_{EF}^2 = 0.8$ , this condition is met.

2. The distributed workload condition is also met.

$$\text{Indeed, } \sum_{h=1}^2 \sum_{first_j=h}^{C_j^{slow}} \frac{C_j^{slow}}{T_j} = \frac{C_1^1}{T_1} + \frac{C_4^1}{T_4} + \frac{C_2^2}{T_2} + \frac{C_3^2}{T_3} \leq 1.$$

3. Now, the admission control has to verify the condition on the sojourn times of the EF flows on nodes common with  $\tau_4$ . This condition is met. For example, the worst case sojourn times of  $\tau_4$  on nodes 1 to 4 are respectively: 8, 17, 25, 36.

4. Finally, the end-to-end response time condition has to be evaluated by the admission control. For example,  $R_{max_4}^{\mathcal{L}}$  (the end-to-end response time of the new EF flow) is bounded by:

$$\underbrace{\sum_{h=1}^4 \left( \sum_{j=1}^4 \left( 1 + \frac{S_m^h + J_{in}^h}{T_j} \right) \cdot C_j^{slow} \right)}_{C_1^1 + C_2^1 + \left(1 + \frac{9+6}{10}\right) \cdot C_3^2 + \left(1 + \frac{9+6}{10}\right) \cdot C_4^2} + \sum_{\substack{h=1 \\ h \neq slow}}^q C_h^h + \sum_{h=1}^q (B^h - 1) + (q - 1)P_{max}.$$

$$C_1^1 + C_2^1 + \left(1 + \frac{9+6}{10}\right) \cdot C_3^2 + \left(1 + \frac{9+6}{10}\right) \cdot C_4^2 = 16$$

7      6      3

The worst case end-to-end response time of the new EF flow is then bounded by 32. Notice that the bound given by the holistic approach is 45, much higher than the bound provided by the trajectory approach.

As part of this article, a simulation tool has been developed for providing the exhaustive solution of a real-time scheduling problem in a network. Indeed, once the different parameters are specified, all the possible scenarios are generated and traffic feasibility is checked for each of them. The result of this simulation is a file containing the real worst case end-to-end response time of each flow and a scenario that leads to this response time. The real worst case end-to-end response time of the new EF flow, provided by the simulation tool, is equal to 26.

It is important to notice that if  $\tau_2$  or  $\tau_3$  are disturbed by a new EF flow not crossing  $\tau_4$ , the real worst case end-to-end response time of  $\tau_4$  will possibly increase while the computed bound will remain the same. For example, if a new EF flow wants to go through the DiffServ domain by node 7, with a period equal to 20 and a processing time of any of its packets equal to 7, it will be accepted by the admission control. Then, the real worst case end-to-end response time of  $\tau_4$  will be equal to 28. If another EF flow having the same parameters wants to go through the DiffServ domain by node 5, it will also be accepted by the admission control. Then, the real worst case end-to-end response time of  $\tau_4$  will increase to 29, while the computed bound will remain equal to 32. Notice that our solution allows a processor utilization for the EF class up to 80% on node 2.

## 8 Conclusion

In this paper, we have proposed a solution to offer deterministic end-to-end real-time guarantees for flows in the EF class of the DiffServ model. The EF class is well adapted for flows with real-time constraints. We have considered sporadic flows that can model voice or video flows. We have determined a bound on the end-to-end response time of any sporadic flow with a trajectory analysis, less pessimistic than holistic analysis and we have compared this bound with the exact value provided by a simulation tool we have designed. The MPLS technology reduces forwarding delays because of simpler processing and allows to indicate in a label the service class of the packet. We have shown how to implement our solution, based on the combination of DiffServ and MPLS. To accept a new EF flow in the DiffServ domain, we have proposed a simple admission control that involves only the EF flows crossing the new flow. Notice that even if the admission control uses the bounds we have established, the resources provisioned for the EF class that are not used are available for the other classes.

## References

1. G. Apostolopoulos, D. Williams, S. Kamat, R. Guerin, A. Orda, T. Przygienda, *QoS routing mechanisms and OSPF extensions*, RFC 2676, August 1999.
2. D. Awdanche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, *RSVP-TE : Extensions to RSVP for LSP tunnels*, Internet draft, August 2001.
3. D. Awdanche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus Wang, *Requirements for traffic engineering over MPLS*, RFC 2702, September 1999.

4. S. Baruah, R. Howell, L. Rosier, *Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor*, Real-Time Systems, 2, p 301-324, 1990.
5. J. Bennett, K. Benson, A. Charny, W. Courtney, J. Le Boudec, *Delay jitter bounds and packet scale rate guarantee for Expedited Forwarding*, INFOCOM'2001, Anchorage, USA, April 2001.
6. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, *An architecture for Differentiated Services*, RFC 2475, December 1998.
7. R. Braden, D. Clark, S. Shenker, *Integrated services in the Internet architecture: an overview*, RFC 1633, June 1994.
8. A. Charny, J. Le Boudec, *Delay bounds in a network with aggregate scheduling*, QoFIS, Berlin, Germany, October 2000.
9. F. Chiussi, V. Sivaraman, *Achieving high utilization in guaranteed services networks using early-deadline-first scheduling*, IWQoS'98, Napo, California, May 1998.
10. L. Georgiadis, R. Guérin, V. Peris, K. Sivarajan, *Efficient network QoS provisioning based on per node traffic shaping*, IEEE/ACM Transactions on Networking, Vol. 4, No. 4, August 1996.
11. L. George, S. Kamoun, P. Minet, *First come first served: some results for real-time scheduling*, PDCS'01, Dallas, Texas, August 2001.
12. L. George, D. Marinca, P. Minet, *A solution for a deterministic QoS in multimedia systems*, International Journal on Computer and Information Science, Vol.1, N3, September 2001.
13. M. Gerla, C. Casetti, S. Lee, G. Reali, *Resource allocation and admission control styles on QoS DiffServ networks*, QoS-IP 2001, Rome, Italy, 2001.
14. J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, *Assured Forwarding PHB group*, RFC 2597, 1999.
15. V. Jacobson, K. Nichols, K. Poduri, *An Expedited Forwarding PHB*, RFC 2598, June 1999.
16. J. Le Boudec, P. Thiran, *A note on time and space methods in network calculus*, Technical Report, No. 97/224, Ecole Polytechnique Fédérale de Lausanne, Swiss, April 11, 1997.
17. F. Le Faucheur, L. Wu, S. Davari, et al. *MPLS support of Differentiated Services*, Internet draft, 2000.
18. A. Parekh, R. Gallager, *A generalized processor sharing approach to flow control in integrated services networks: the multiple node case*, IEEE ACM Transactions on Networking, Vol.2, N2, 1994.
19. D. Raz, Y. Shavitt, *Optimal partition of QoS requirements with discrete cost functions*, INFOCOM'2000, Tel-Aviv, Israel, March 2000.
20. V. Sivaraman, F. Chiussi, M. Gerla, *Traffic shaping for end-to-end delay guarantees with EDF scheduling*, IWQoS'2000, Pittsburgh, June 2000.
21. V. Sivaraman, F. Chiussi, M. Gerla, *End-to-end statistical delay service under GPS and EDF scheduling: a comparaison study*, INFOCOM'2001, Anchorage, Alaska, April 2001.
22. K. Tindell, J. Clark, *Holistic schedulability analysis for distributed hard real-time systems*, Microprocessors and Microprogramming, Euromicro Journal, Vol. 40, pp. 117-134, 1994.
23. M. Vojnovic, J. Le Boudec, *Stochastic analysis of some expedited forwarding networks*, INFOCOM'2002, New York, June 2002.

# Measuring and Communicating Component Reliability

John D. McGregor, Judith A. Stafford, and Il-Hyung Cho

Clemson University, Tufts University, Clemson University  
{johnmc, ihcho}@cs.clemson.edu, jas@cs.tufts.edu

**Abstract.** Much of the research on component-based software engineering assumes that each component has a single service. This simplifies analyses but it is a significant departure from actual components. This paper reports on an investigation of the feasibility of using design constructs as a means of treating several methods as a single unit. We introduce the CoRe Method for measuring and communicating component reliability based on grouping services according to the various roles a component can fulfill. Grouping the services of a component into a few sets satisfies the goal of simplicity while still providing the designer with a more realistic and useful model of component reliability. The CoRe method is described and results of a preliminary experiment in its use are reported.

## Introduction

The reliability of a software system expresses the probability that the system will be able to perform as expected when requested. This probability is interpreted in the context of an expected pattern of use and a specified time frame. A system's reliability might be reported as:

**System X has a reliability of .99 when used in normal mode during an eight-hour period of operation.**

Normal mode refers to a specific system mode that is defined by a pattern of inputs to the system. This may include specific selections from menus or certain characteristics of data in the input files.

Most components have multiple entry points just as most systems have multiple menu selections. When a system is being developed to meet the requirements of a specific customer it is reasonable to expect that the customer can identify expected patterns of use. However, when software is being designed to be used in many different ways by different organizations as is the case when software components are being developed, predicting usage patterns becomes a very hard problem.

The long-range goal of this research is to define a reliability measurement technique that provides the information necessary to support the prediction of assembly reliabilities from the reliabilities of the components that form the assembly. The patterns of use for a component will vary with the context in which it is placed and the expected use of the system that is comprised of the components. Because reliability measurement depends on identification of patterns of use and because the reliability of the assembly depends on the reliability of its constituent components an important aspect of our research is to develop a context independent means for reporting infor-

mation that can be used to calculate the reliability of a component once its context is known. As mentioned in the previous paragraph, software components have many possible entry points, each of which might be used independently of the others and contribute independently to the overall behavior of the assembly. Therefore, it seems unreasonable to consider the component as a whole when calculating assembly reliability. We observe that combinations of entry points are often used in combination to perform some task. That is they participate together in some role that the component plays.

The hypothesis of this specific investigation is that the role a component plays in a specific protocol is a useful unit upon which to base a reliability measure. We use *protocol* here in the sense that it is used in the real-time extension to UML[1], in the architecture description language Wright[2], and in our previous work[3]. A protocol is a sequence of service invocations between a set of components. A protocol accomplishes a specific task through the interaction of these components. We begin our definition of component reliability by measuring the ability of a component to perform its role in a protocol as expected.

A fundamental assumption of this work is that components are designed to satisfy roles in various standards or design patterns. We assume that no component is developed in the absence of some rationale for its existence. That rationale might be that the component is intended to play the role of server in a communication protocol. The component would be designed to transform data from the transport format into a computation format.

The contribution of this paper is a the *Component Reliability* measurement method (*CoRe*) for measuring and representing the reliability of a component that is useful in describing components that are intended for use by other developers. The method provides a technique for decomposing the specification of the component into logical pieces about which it is possible to reason. The method then provides a context in which the reliabilities of these logical pieces can be combined to match the complete use a developer defines for the component in a specific system. For example, a component may play the role of server in one interaction while playing the client role in another protocol. The effective reliability of the component is a combination of the reliabilities of the two roles.

In the next section we provide background on components and reliability sufficient to understand the investigation. The following two sections define the method and provide a detailed example respectively. We conclude with a look at how this approach fits into a compositional reliability prediction method.

## Background

### Components

The work described in this paper conforms to the standard properties of a *component* defined by Szyperski [4]:

- unit of independent deployment,
- unit of third-party composition,
- no persistent state,

but reliability measures are not directly affected by variations in these properties. We assume that a component supports multiple public services that have been specified in one or more interfaces. Each interface describes a set of services that relate to some common purpose such as being a participant in a design pattern.

A component is connected to other components in conformance to an architecture. In the architecture description language Wright[2], the connector is a first class object that defines a protocol and contains a set of role definitions needed for the protocol as shown in Figure 1. Each component defines a set of ports that identify the roles the component implements. Any component that implements a role has the potential to be attached to the role via a specific port. The protocol is characteristic of the design such as a client/server protocol.

Both the port and role descriptions provide sufficient information to check the compatibility of a specific port and specific role. A port of a component is compatible with a role if the port can be substituted for the role with no noticeable difference in behavior. In addition to the usual type compatibility notion, a protocol usually imposes a partially-ordered sequence on the service invocations provided by the roles involved in the protocol.

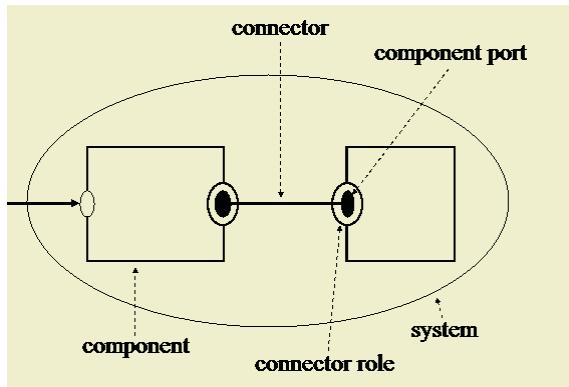


Fig. 1.

## Reliability

The reliability of a software product is usually defined to be “the probability of execution without failure for some specified interval of natural units or time”[5]. This is an operational measure that varies with how the product is used. Reliability of a component is measured in the context of how the component will be used. That context is described in an operational profile.

Reliability of a piece of software may be computed or measured based on experience with use or testing of the software. If the software has not been built yet, its reliability can be predicted from a structural model and the reliabilities of the individual parts that will be composed to form the software. There is error associated with this technique due to emergent behaviors, i.e. interaction effects, which arise when the components must work together.

If the software is already assembled, the reliability can be measured directly. The measurement is taken by repeatedly executing the software over a range of inputs, as

guided by the operational profile that was identified for the assembly. The test results for the range of values are used to compute the probability of successful execution for a specific value. The error associated with this approach arises from two sources: the degree to which the actual operation deviates from the hypothesized operation assumed in the operational profile and any error resulting from the mechanics of measuring the output of the system.

## Two Types of Confidence

Reliability measurements are based on testing and are therefore subject to uncertainty. It is therefore necessary to use confidence intervals to measure how certain we are about the measurement we have made. It is important to recognize that the confidence interval does not necessarily describe how confident we are that we have measured the correct quantity. For example, we might choose parameters for three test cases, all of which result in the same ten lines of code being executed. Since the same lines of code are executed each time, there will be very close agreement among the three measures. As a result, our confidence in the accuracy of the measurement will be high even though choosing other parameter values might have led to a very different result. It is the variance among the measurements, or the lack thereof, which gives confidence that we have made an accurate measure of something.

The operational profile for a role gives confidence that the reliability test suite will measure the correct quantity. The more closely the test suite resembles the actual use of the functionality described in the role, the more accurate the resulting reliability value is. If portions of the software are not executed, it is because the role does not use that software and it is not pertinent to the measurement. For an indepth discussion of the use of confidence intervals in software reliability measurement see Trivedi[6]. CoRe, which is described in the next section, uses both an operational profile for the unit being measured and a confidence interval about the measured value to report the reliability for a component role. In this way we have a high confidence that the measurement is an accurate value for the correct quantity.

## CoRe Method

The **Component Reliability** measurement method (CoRe) is a technique for making empirical measurements of the reliability of a software component. The technique does not necessarily require access to, or knowledge of, the source code and, as such, can be used by third party certifiers or the assembly developer[7]; however, the purpose of CoRe is to aid the component developer in providing information to potential acquirers of the component.

We assume the public specification of the component, as exposed through the interfaces, i.e. ports, that it implements, identifies the various roles with which the component is intended to be compatible. Each role is implemented by a group of methods. CoRe uses “role” as the unit for which reliability measurements are made. The definition of each role is used to create an operational profile. The role’s operational profile is then used to create a set of test cases that are used to measure the reliability.

## Method Definition

The definition of reliability given in the previous section has three significant parts:

1. A probability
2. An operational profile that defines context
3. Duration of operation defined in some natural units

We consider them in reverse order of appearance in the definition.

We discuss these parts of the definition in terms of the example we will be using for a set of experiments. The example component is an SSH client. This component can be configured to act as an SSH1 client or an SSH2 client. We selected this component because the SSH state machines are well-defined and the protocols between clients and servers are clearly specified.

## Natural Units

The reliability of a complete system, particularly a continuously operating system such as a telephone switch, is usually defined in terms of execution time. The duration of a reliability test run is then chosen to correspond to some time period of interest, such as an eight-hour workday. Further, the execution time can be translated into calendar time if that is a more meaningful level for the analysis. This can be used to report reliability values for a system that has a natural period of a week's worth, a month's worth or even a year's worth of operation.

For an individual component, the duration of execution of a specific service may be very brief. Some components might only execute for a very brief time during a lengthy system execution time. Associating a duration of eight hours with a component that only executed a few milliseconds during that period is not representative. When the component is placed in a system that invokes it more often, its contribution to the system reliability may vary markedly from that observed in the original system.

Although the reliability of an individual component could be expressed in terms of number of invocations of individual services, this seems too fine-grained to be useful. Architects are unlikely to know exactly how many times a service of a component will be invoked and thus would not carry out their analysis of a system at that level. CoRe reports reliability values based on the probability that a component will execute its role in a protocol successfully. For example, our SSH client might require multiple invocations of the decryption service of the SSH component, but the architect is concerned with whether a file transfer is completed or not. The exact number of invocations of the decryption algorithm is not important to the analysis of the overall reliability.

## Operational Profile

Reliability is an operational concept. A piece of software's reliability value will vary depending upon how that software is used. Suppose a component has five services, four of which always succeed and one of which always fails. If all services are used at an equal rate, the system is 80% reliable over an aggregate of many uses. But if the service that always fails is never used, the system is 100% reliable.

An operational profile describes the frequency with which we expect each service to be invoked. In CoRe an operational profile is created for each role in each protocol in which the component is designed to participate. We assume that the services for a particular role are contained in specific interfaces implemented by the component.

The SSH Client component implements separate interfaces for SSH1 and SSH2 types of connections.

A state machine and a set of frequency tables are used to describe the operational profile for a role. The state machine defines the sequence in which the services of the component can be invoked. For the SSH client, the basic state machine defines a sequence of (1) establish transport layer; (2) authenticate client to server; and (3) connect to desired services.

A frequency table is used to capture the relative frequency with which each service is invoked in one complete cycle of the protocol. There may be multiple frequency tables particularly if there are multiple paths through the state machine. Even a single path may have cycles that permit multiple frequency values for some services. For the SSH client, the transport layer is established once per session. Authentication occurs once per client login. The connection is used an indefinite number of times for a variety of services. This last action has the greatest impact on reliability since it is the action taken most frequently.

A system may have many operational profiles corresponding to different users and it may have many profiles for one user if there are different modes of the system. Similarly, a component may be able to play many roles in many protocols and thus an operational profile is created for each role. CoRe reports a reliability value for each role. It is possible to identify users who only use the SSH client for SSH1 connections, others only for SSH2 connections, and others who use both. It is also possible to identify users who primarily use the SSH client for single operations in which case all three services are used with the same frequency and those users who maintain a connection for a long time using several services with a single transport/authentication action.

### **Probability of Successful Execution**

The probability of successful execution is measured by repeatedly operating a system according to the selected operational profile, i.e. selecting inputs according to the frequency constraints of the profile, for the specified unit of time. The reliability is computed by measuring the percentage of those executions that terminate successfully.

For a component that will be used in a variety of contexts, a reliability value should be reported for each operational profile so that the architect can compute the effective reliability that will be observed in a particular system. For example, a component is provided that plays a role in each of five protocols. The reliability portion of a component datasheet contains the role reliabilities and abbreviated role descriptions as in Table 1.

The *effective* reliability of the component in a particular deployment is the reliability that the architect will experience with a specific component used in a specific manner. It is computed by multiplying the relative frequency for a role, as anticipated by the architect, and the measured reliability of the component in that role. These values are combined by summing to give the effective reliability of the component in all of its roles.

Suppose the architect intends to use the first, third, and fifth roles with equal frequency. The architect's reliability analysis worksheet includes a computation that looks something like the right two columns of Table 1 for each component defined in the architecture.

**Table 1.** Combining role reliabilities.

Role	Description	Reliability by role	Relative Frequency	Contribution
A	Provides basic computation	.9	0.333	0.30
B	Provides graphing...	.91	0.0	0.0
C	Provides database access ...	.92	0.333	0.31
D	Provides security...	.93	0	0.0
E	Provides transaction ...	.94	0.333	0.31
Component Reliability				0.92

This value can be used in a high-level reliability analysis for the system, which uses the topology of the components in the architecture to compute estimated system reliability. More detailed system analyses may use the individual reliability values.

## CoRe Process

CoRe measures and communicates reliability values for a component. In this section we present a detailed outline for applying CoRe given a component and its documentation.

### Establish the Context for Measurements

Establish the confidence you wish to have in the accuracy of the reliability value. The typical value is 95% or higher.

Establish exactly how “operating as expected” will be measured. This may be as simple as obtaining the correct numerical value or the result may be more subjective and require expert evaluation.

### Identify the Component’s Roles

Identification of the component’s roles comes from the documentation of the component. The designer might have listed the roles and identified the services that participate in those roles. The reliability test plan identifies each of the roles and for each role the services that implement the role. The documentation for the SSH client states that the client provides a “complete” implementation of the SSH1 and SSH2 protocols. The SSH standards document defines the basic roles and provides the definition of each of the services.

### Establish an Operational Profile for Each Role

The component’s operational profile describes the relative frequency with which each service is used within the role. The reliability test plan defines how often the test suite

should invoke each service of the role. Each test case will be one complete performance of the role. In that performance some of the services may be invoked multiple times. For cases where the number of times a service  $s$  may be called depends on some data element, that element becomes a quantity to vary from one run to another. For example, if a transaction consists of a number of actions, depending upon the user's need, the reliability test suite should include cases that systematically vary the number of actions.

### **Construct a Reliability Test Suite for Each Role**

For each role, a test script is created that obeys the constraints of the role. The constraints usually include the state transitions that are possible. Other constraints include the types for the parameters on each service. We assume no access to the source code so the analysis is limited to the public interface documented in the role description. An analysis of each parameter type in the role description leads to a set of partitions of values from the type. Each partition contains values for which we believe the component will behave the same. The test suite includes test cases that (1) test each possible ordering of service invocations and that (2) adequately sample over the parameter partitions for each service.

A thorough analysis of the SSH client and the SSH protocol indicates that, since the client runs on top of standard TCP implementations, the test cases do not need to cover multiple platforms. Also the number of orderings of invocations is very limited since establishing the transport layer followed by authentication must occur in that order. The mixture of service invocations after the connection is established is the main source of variation among test cases.

This analysis produces far more test cases than can realistically be administered in a test run. For a given test run, test cases are sampled from the test suite and applied to the component.

### **Apply Each Test Run in an Appropriate Environment**

A test run will cover a complete performance of the protocol. For example, if the protocol is a transaction, a test run would extend from transaction start to transaction commit or transaction rollback. Multiple test runs will be conducted where one run differs from another by varying input parameters.

### **Analyze the Results and Compute the Reliability**

Each test case is evaluated and marked as either passed or failed. Each failure case is evaluated to determine whether the environment produced the failure or whether the component is responsible. The reliability computation uses the number of success and the number of failures to compute the reliability.

### **Expand Test Suite to Reach the Level of Confidence Desired**

Once the analysis is completed, if the level of confidence has not been reached, additional test cases may be created and executed. The coverage criteria given in step 4.3 provide direction on how to effectively expand the test suite to cover additional ground.

## Experimentation

We have conducted a set of experiments using the public domain secure shell component described earlier in the paper. That component was viewed as having two roles from the user's point of view. The first role is providing telnet service and the second is providing ftp service. The component was also viewed as providing these services using the SSH1 and the SSH2 protocols.

### Operational Profiles

Creating the operational profiles for a component is made easier in our approach but it still requires a logical derivation of the profile from either actual usage information or from deductions about how the component will be used. We created a profile for each of the two roles for the component using deductions about the roles.

Ftp is a file transfer protocol. The two attributes of that transfer which can be controlled are the size of the file to be transferred and the content of the file. Since there was no reason to expect that one of the nine file types would be used more than another, we used a uniform probability distribution across the nine types as shown in Table 2.

**Table 2.** Operational profile for ftp role.

ftp	Small file	Medium file	Large file
Text	11.1%	11.1%	11.1%
Binary	11.1%	11.1%	11.1%
Proprietaryformat	11.1%	11.1%	11.1%

Telnet allows the client to perform commands on the server machine. One dimension of this is the accuracy of the input given during a telnet session. We selected standard values for error rates for two classes of users: novice and expert. The second dimension was the type of action the client invokes on the server. Commands produce some amount of output from very little input. Executing a program such as a text editor has a high level of two-way communication. Once again we used a uniform distribution across nine cases as shown in Table 3.

**Table 3.** Operational profile for telnet role.

telnet:	Novice –	Expert –
	10% error rate	1% error rate
Command line – short query	12.5%	12.5%
Command line – run application	12.5%	12.5%
Editor	12.5%	12.5%
Mail	12.5%	12.5%

## Experimental Results

Tables 4 and 5 show the results of the reliability testing. (No failures were experienced in testing the ftp.) Table 4 shows differing numbers of failures for differing cells. The combined result of the reliability testing, Table 5, shows that a user of this component will experience higher reliability from the ftp service than from the telnet service.

**Table 4.** Reliability results for the telnet role.

telnet:	Novice – 10% error rate	Expert – 1% error rate
Command line – short query	100%	100%
Command line – run application	100%	100%
Editor	90%	99%
Mail	90%	99%

**Table 5.** Combined reliability results for SSH component.

	SSH1	SSH2	
ftp	100%	100%	100%
telnet	97.25%	97.25%	97.25%
	98.625%	98.625%	

The exact effective reliability that will be experienced by the user of the SSH component will depend upon the use to which it is put. By reporting separate reliability values for the ftp and telnet roles, the architect can better estimate the reliability of their specific overall system.

## Integrating CoRe with a System Reliability Estimation Technique

The purpose of CoRe is to provide component information that can be used by an architect to estimate the reliability of a system. The system comprises a set of components whose reliabilities have been evaluated using CoRe. In this section we describe the use of CoRe with Musa's system reliability estimation technique[5].

The issues surrounding the integration of CoRe and a system estimation technique include:

- How to account for interactions among services
- How to model components with multiple roles

### Interactions among Services

The multiple services within a component often interact through shared variables local to the component. CoRe accounts for one level of interaction by recognizing that a role is implemented by services in a single interface<sup>1</sup>. It is natural that services that implement different portions of a single interface specification would have state in common. CoRe handles this by measuring the reliability of the component's role in a protocol. The effects of the interactions are accounted for in the measurement. For the purposes of this paper we will assume this is the only type of interaction present. This

---

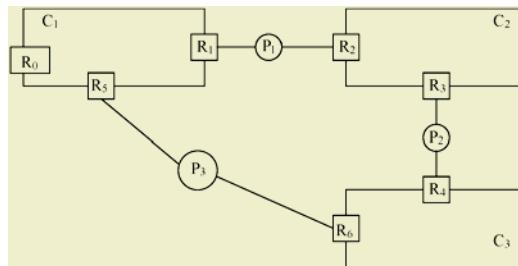
<sup>1</sup> Where there are multiple interfaces per role we assume that all of the services of each interface are involved in the role.

is the common assumption by most reliability modeling techniques: failures are independent.

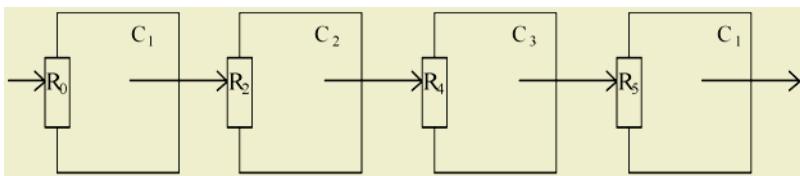
The second level of interaction, state shared by services in multiple interfaces in separate roles, is more problematic. These interactions often cannot easily be described in a component's specification. One possible approach would be a protocol-level state machine. Another possibility is the use program dependence analysis to identify potential internal conflicts[8]. Measuring and communicating this level of interaction is future work.

## Modeling Multiple Roles

Consider the three components in the architecture drawing in Figure 2. If we consider the operational relationships and place the components in the control flow format compatible with Musa's modeling approach we get the pipeline shown in Figure 3. The  $C_1$  component appears in two places in the model. The two instantiations of  $C_1$  are possible because of the assumption in the previous section. Actions on the first occurrence of  $C_1$  have no impact on the second occurrence of  $C_1$  even though the two occurrences in the model are realized by a single object in the implementation.



**Fig. 2.** Components with multiple roles.



**Fig. 3.** Pipeline format.

The reliability computation for the pipeline shown in Figure 3 is given by:

$$C_{1R_0} * C_{2R_2} * C_{3R_4} * C_{1R_5}$$

The reliability of each interface is independent of the others.

## Related Work

The study of compositional reliability has attracted the interest of several researchers in recent years. It is important to recognize that the problems being explored by these researchers are fundamentally different than those that have been dealt with by over the past decades by members of the software reliability of community (e.g. [5][6]). This difference arises from the fact that earlier work was based on the assumption that all components were being developed under the control of the assembler and would be incrementally improved until the desired system reliability could be attained. Recent work on compositional reliability is focused on problems associated with integrating pre-existing components. There are many issues to be addressed before a solution to the general problem of predicting assembly reliability will be solved. Current work covers several of these issues. Reussner and Schmidt suggest the use of parameterized contracts as to support automated composition and property prediction[9]. Hamlet et al. have devised a method for propagating operational profiles of appropriately annotated components along execution paths of assemblies[10]. Yacoub et al. identify the need to include consideration of the reliability of connectors in any compositional reliability model and propose a model that does so[11]. Our work complements these works in that it is attaching an additional aspect of the problem.

## Future Work

The strength of the CoRe method is the ability to control the level of granularity for which a reliability value is reported. However, additional work is needed on the CoRe method to better define the technique for identifying the operational profile for the selected granularity.

The next step in the maturation of CoRe is to integrate this work into the driving work on prediction-enabled components. The CoRe method is a fundamental element in an effort to construct a Prediction-Enabled Component Technology (PECT) that predicts the reliability of a component assembly from the reliabilities of the individual components. The Reliability PECT works with components that have been evaluated using the CoRe method. The PECT provides an analysis technique that models the proposed assembly, selects the appropriate roles and corresponding reliability values for each component used in the assembly, and predicts the reliability value of the assembly.

A number of issues concerning the composability of reliability predictions remain. How are the confidence intervals about the individual reliability values combined to give a bounds to the reliability of the composition of two components? What is the effect on the assembly reliability of composing two components where the output of the initial component is a subset of the expected input of the second component? Finally, a fundamental issue is how to account for dependent failures.

## Conclusion

This work is exploring how to provide useful information about reliability to acquirers of components. Rather than provide a single value for the entire component, we

provide reliability information about each role that the component is intended to support. The acquirer can then compute the effective reliability they would experience given their intended use of the component. The intention is to provide accurate and useful information about reliability in support of component commerce and prediction of assembly reliability.

## References

1. B. Selic and J. Rumbaugh. *Using UML for Modeling Complex Real-Time Systems*, Rational Corp., 1998.
2. R. Allen and D. Garlan. "A Formal Basis for Architectural Connection," *ACM Transactions on Software Engineering and Methodology*, 1997.
3. I. Cho and J. D. McGregor. "Component Specification and Testing Interoperation of Components," IASTED 3rd International Conference on Software Engineering and Applications, Oct.1999.
4. C. Szyperski. *Component Software: Beyond Object-Oriented Programming*, Boston: Addison-Wesley, 1998.
5. J. Musa. *Software Reliability Engineering*, New York: McGraw-Hill, 1998.
6. K. S. Trivedi. Probability and Statistics with Reliability, Queuing and Computer Sciene Applications, Second Edition, Wiley Interscience, 2002.
7. J.A. Stafford and K.C. Wallnau. "Is third party certification really necessary?", Proceedings of the Fourth ICSE Workshop on Component-Based Software Engineering, Toronto, CA, May 2001, Pgs 13-19.
8. J.A. Stafford and A.L. Wolf. "Annotating Components to Support Component-Based Static Analyses of Software Systems." *Proceedings of the Grace Hopper Celebration of Women in Computing 2000*, Hyannis, Massachusetts, September 2000.
9. R.H. Reussner, I.H. Poernomo and H.W. Schmidt. Reasoning on Software Architectures with Contractually Specified Components.  
In A. Cechich and M. Piattini and A. Vallecillo, editors, *Component-Based Software Quality: Methods and Techniques* to appear. Springer-Verlag, Berlin, Germany, 2003.
10. D. Hamlet, D. Mason, & D. Woit. "Theory of Software Reliability Based on Components." *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2002)*. Toronto, Canada, May 2001. Los Alamitos, CA: IEEE Computer Society Press, 2001.
11. S. Yacoub, B. Cukic, and H. Ammar. "Scenario-Based Reliability Analysis of Component-Based Software," in *Proceedings of Tenth International Symposium on Software Reliability Engineering (ISSRE99)*, Boca Raton, Florida, November 1999, Pgs. 22-31.

# Toward Component-Based System: Using Static Metrics and Relationships in Object-Oriented System

Eunjoo Lee<sup>1</sup>, Byungjeong Lee<sup>2</sup>, Woochang Shin<sup>3</sup>, and Chisu Wu<sup>1</sup>

<sup>1</sup> School of Computer Science and Engineering, Seoul National University  
Seoul, Korea

{banny, wuchisu}@selab.snu.ac.kr

<sup>2</sup> School of Computer Science, University of Seoul, Seoul, Korea  
bjlee@venus.uos.ac.kr

<sup>3</sup> Dept. of Internet Information, Seokyeong University, Seoul, Korea  
wcshin@imail.skuniv.ac.kr

**Abstract.** Object-oriented development had not provided extensive reuse and computing infrastructures are evolving from mainframe to distributed environments, where object technology has not led to massive development of distributed systems. However, component-based technology is considered to be more suited for distributed system development due to its granularity and reusability. In this paper, we present a process including the metrics and guidelines that can be applied to reengineer object-oriented systems into component-based systems. Components are created based upon generalization and composition relationships between classes. Resulting components are refined by using the metrics we propose.

## 1 Introduction

The increasing complexity and shorter life cycle of software has made it necessary to reuse software. Reuse has increased the reliability of software applications and made it efficient to develop and maintain current software. Object-oriented technology has triggered the software reuse. However, object-oriented development has not led to extensive reuse since classes are too detailed and specific, and further they had to be bound with an application either at compile-time or link-time [1].

In the past, a ‘legacy system’ has been regarded as a system that was implemented with procedural languages, such as COBOL, FORTRAN, C, etc. However, computing environments are evolving to distributed systems running applications that can be developed using several different languages that may be operating on several heterogeneous platforms. Many modern systems are component-based in these environments, that is, they focus on existing components and the creation of new components.

Components are regarded as being more coarse-grained compared to traditional reusable artifacts, such as objects, and are intended to provide a high-level representation of the domain. Fine-grained traditional artifacts prevent managers from effec-

tively working with these artifacts. However, the coarse-grained domain-oriented approach used for components allows managers to identify the components that satisfy their business requirements, and to assemble them into full-scale applications [2].

Components have greater granularity and have more domain-specific aspects than objects. Therefore, components can be used more effectively than objects in a distributed environment. Also they are better suited for reuse.

Thus, the interpretation of a ‘legacy system’ has changed to include applications based upon a stand-alone object-oriented system with the advent of these environments. In order to provide extensive reuse and reengineer object-oriented systems into systems suitable for a distributed environment where many systems make use of components, we should derive reusable components from classes in object-oriented systems and change object-oriented systems into component-based systems.

Several studies have investigated reengineering a legacy system into an object-based distributed system [3, 4, 5]. The systems written in procedural language have been reengineered in these studies. An approach to identify the components from an analysis level object model has been proposed [2], but the approach has not provided a complete methodology for reengineering an object-oriented system into a system that consists of components [2].

In this paper, we present a process to reengineer an object-oriented system into a component-based system. First, we identify the basic components based upon relationships between classes from program source code. Next, we define component metrics, which are useful to derive components that can be used as cohesive functional units in a distributed environment, to assess the quality of components. Finally, we apply these metrics to refining basic components.

The rest of this paper is organized as follows. Section 2 shows related work and their inherent problems. In Sect. 3, we describe a process to generate components from object-oriented classes. In Sect. 4, we apply our process to an example of object-oriented system and show reengineering the example into a component-based system. Finally, Sect. 5 provides some conclusions and suggestions for future work.

## 2 Related Work

A semiautomatic evolutionary migration methodology for legacy systems has been proposed, which has produced an object-based distributed system [3]. Although components have been defined as “a set of objects wrapped together with an interface” in this study, this work has reengineered a procedural system, not an object-oriented system. Also, there have been only a few studies done on the properties of the components, such as quality.

An approach to identify components from an object model that represents a business domain has been introduced [2]. This approach has adopted a clustering algorithm, a predefined rule and a set of heuristics. A hierarchical and agglomerative clustering with static and dynamic relationship strength between classes was used in this study. Static relationships were derived from association between classes and dynamic relationships were derived from use cases and sequence diagram. Super-

classes are replicated to the components containing one or more subclasses in order to enhance clustering solutions. This approach has applied a set of heuristics automatically executed by a system or manually performed by a designer to refine the solution obtained from the clustering algorithm. However, this approach has not provided any description of a component interface. This approach, however, is inefficient and not useful because a parent class must be replicated into every component if the component has at least one child class of the parent class.

A component identification method that involves component coupling, cohesion, dependency, interface, granularity, and architecture has been proposed [6]. Since this method requires some analysis and design information, use-case diagram, sequence diagram and collaboration diagram, etc like the approach [2], this method is expected to be used in component-based development (CBD).

A set of reusability metrics to be iteratively applied during the design and implementation parts of the software lifecycle has been introduced to guide the production and identification of reusable components [7]. The goal of this study is to increase domain-specific reuse and this approach requires developers to subjectively categorize classes, identify component boundaries, and specify where components are related. So it is necessary for software designer to have a solid understanding of application domain.

A quantitative approach to the modularization of object-oriented systems uses cluster analysis, which is a statistical method of grouping classes [8]. The dissimilarity between classes in cluster analysis is based on their relative couplings classified according to a taxonomy scheme, where the categories are assigned weights. Modules can be components in this study, but the module couplings consider only a static relationship between classes and not an amount of information transferred between classes. Moreover, the final number of modules must be set in advance before clustering.

### 3 Reengineering Process

#### 3.1 Deriving Design Information from Source Code

We assume the following:

- Generalization relationship between classes: We say that class X and Y have a generalization relationship or that class Y is a child of class X (class X is a parent of class Y) when class X is a direct parent of class Y in an inheritance hierarchy. A leaf class is a class that has at least one parent class but does not have any child class.
- Composition relationship between classes: We say that class X and Y have a composition relationship or class X has class Y as a composition member class, when class X has class Y as a member variable, class Y has a strong attachment to class X and the same life cycle as class X's.

- Use relationship between classes: We say that class Y uses class X when:
  - Any method of class Y gets the instance x of class X as its parameter,
  - Class Y has the instance x of class X as its member variable except for the composition relationship,
  - In any method of class Y, an instance of class X is declared and used.
- Dependency relationship between components: We say that component Y is dependent on component X if component Y uses services that component X provides.
- An object-oriented system S is composed of interacting classes
  - $S \subset UC$
- A reengineered system  $S'$  consists of a set of components.
  - $S' \subset UCOMP$
- UID is the universe set of unique identifiers.
- DTYPE is the universe set of all data types.
  - $DTYPE = PTYPE \cup ETYPE \cup UC$
  - PTYPE is the set of all primitive types such as ‘integer’, ‘real’, ‘bool’, ‘string’, ...
  - ETYPE is the universe set of all enumeration types.
- SIG is the universe set of method signatures.
  - $SIG = \{ <name, s_1 \times \dots \times s_n, s> \mid name \in UID, s_1, \dots, s_n, s \in DTYPE \cup \{\phi\} \}$
  - $\phi$  is an empty value.
  - $SORTS(x) = \{ s, s_1, \dots, s_n \}$ , where  $x = <name, s_1 \times \dots \times s_n, s>$ .
- UC is the universal set of classes. Each class  $c$  of UC is defined as a tuple  $<name, ATTRSET, MSET, GEN, CP, USE>$ 
  - $name \in UID$
  - $ATTRSET \subset DTYPE$
  - MSET is a set of  $<name_M, sig, body>$  representing methods.
    - $name_M \in UID$
    - $sig \in SIG$
    - $body$  is the method’s executable (source) code.
    - $SIG(m)$  is a function projecting a method  $m$  to its  $sig$  element.
    - $SIGLIST(m) = SORTS(SIG(m))$
  - GEN is a set of parent classes
    - $GEN = \{ p \mid p \in S, c \text{ inherits } p \}$
  - CP is a set of composition member classes.
    - $CP = \{ d \mid d \in S, \text{composition}(c, d) \}$
    - $\text{composition}(x, y)$  is true if class  $x$  has class  $y$  as a composition member class, otherwise false
  - USE represents a set of classes that is used by  $c$ 
    - $USE = \{ <u, m> \mid u \in S, m = \phi \wedge m \in MSET(u) \}$
    - $MSET(x)$  is a function projecting a class  $x$  to its MSET element.
- UCOMP is the universal set of components. Each component  $q$  of UCOMP is defined as a tuple  $<name, INTERFACES, CSET, COMPDEF>$ .

- $name \in \text{UID}$
- INTERFACES represents all interfaces that  $q$  provides. Each interface  $i$  of INTERFACES is defined as follows:
  - $i = <name_i, sig>$ , where  $name_i \in \text{UID}$ ,  $sig \in \text{SIG}$
- CSET is a set of constituent classes of  $q$ .
  - $\text{CSET} \subset \text{UC}$
  - $\text{CSET}(x)$  is a function projecting a component  $x$  to its CSET element.
- COMPDEF is a set of components that  $q$  is dependent on
  - $\text{COMPDEF} \subset \text{UCOMP}$

Based on the above, we formally describe a system with tuples and sets. This formal model reduces a possibility of misunderstanding a system and enables operations to be executed correctly.

### 3.2 Identifying Basic Components

We identify basic components as an intermediate result in our process that is generated based upon the generalization and composition relationships.

#### Step 1 Basic Component with Composition Relationship

The ideal scenario is one in which the cohesion within a component is maximized and the coupling between components is minimized [2]. Classes with a compositional relationship can be used as one functional unit. The separation of these classes into other components will result in low cohesion within a component and high coupling between components. Therefore, we cluster them into a single component as follows:

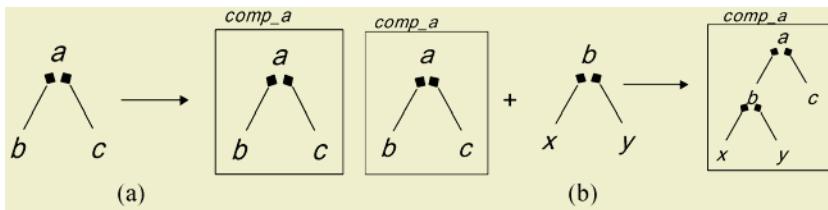
For each class in a system that has any composition member class:

1. If class  $a$  is not contained in any composition basic component, a component  $comp\_a$  is created and with class  $a$  and the member classes of class  $a$  are copied to  $comp\_a$ . (Fig. 1(a)).
2. If class  $b$  is already contained in a component  $comp\_a$ , the composition member classes of class  $b$  are copied to  $comp\_a$ . (that is, they are contained in  $comp\_a$ ) (Fig. 1(b)).

A solid line with a filled diamond represents a composition relationship like UML notation. In Fig.1(a) and Fig. 1(b), class  $a$  has class  $b$  and  $c$  as its composition member class.

#### Step 2 Deleting Needless Classes (I)

If any class contained in a component is not called by other components or other classes and has no child class, it is deleted. If it is not deleted, it may result in unnecessarily creating generalization basic components at the next step.

**Fig. 1.** Composition relationship.**Step 3** Basic Component with Generalization Relationship

If there are inheritance relationships between classes, then it is more appropriate to place those classes in the same component because of the strong relationship between them. If parents and children classes were distributed across components, it would result in an increase in dependency (coupling) between components [2]. However, if a system is composed of several classes with a generalization relationship, it is not practical to make a component for every leaf class, copying all of its parents into the component. On the other hand, if all leaf classes that have common parents and their parents are copied into a single component, that component will be too complex.

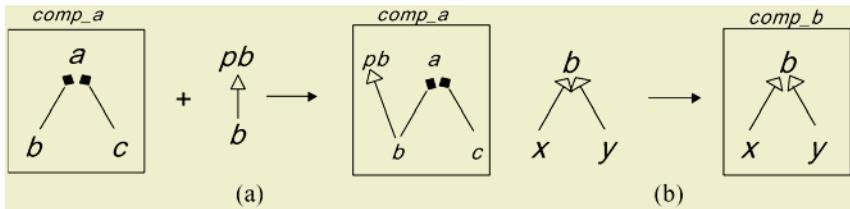
We classify classes with a generalization relationship into two types: concrete and abstract. For each group of generalization-related classes, a new component is created, all abstract classes in the group are copied to the component and all concrete classes in the group are moved to. In this process, “A class is copied” implies that the original of a copy remains in  $S$  and it is possible for the original to be an abstract parent of other classes. “A class is moved” implies that the class is removed in  $S$  and it is no longer used in Step 3.

In order to avoid inheritance between components, which is an undesirable relationship, if a class in a component has an ancestor class, which is not contained in the component, the ancestor is copied to the component (Fig. 2(a)). This process can be described as follows:

For all classes and composition basic components from Step 1:

1. If a class ( $b$  in Fig. 2(a)) in a component ( $comp\_a$  in Fig. 2(a)) is generalization-related with other classes, which are not contained in the component, the ancestors of the class are copied into the component (Fig. 2(a)).
2. If a class ( $b$  in Fig. 2(b)) has one or more descendant classes, a component ( $comp\_b$  in Fig. 2(b)) is created and then the class and all descendant classes are copied or moved into  $comp\_b$  according to their type; abstract classes are copied and concrete classes are moved. (Fig. 2(b)).

An arrow with empty triangle represents generalization relationship like UML notation. In Fig. 2(a), class  $pb$  is a parent of class  $b$ .



**Fig. 2.** Generalization relationship.

#### Step 4 Deleting Needless Classes (II)

If a class is copied into one or more components at least once, it is deleted to eliminate class redundancies and unnecessary refining operation described in the following subsection.

Following the above steps to identify basic components controls the complexity of components by restricting classes contained in each component and improves the efficiency of the reengineering process into component-based systems by classifying classes with a generalization relationship into two types.

### 3.3 Refining Components

In this subsection, we define component metrics including connectivity strength (*CS*), cohesion, and complexity and refine classes and basic components by grouping related classes and components into a larger component using the metrics. After refining components, a component interface for each component can be created

**Component Metrics.** Component *CS* and cohesion indicate the intensity between components and the intensity between elements of a component, respectively. A component complexity tends to affect the understandability and adaptability of a component.

The *CS* value represents how closely a component is connected with other components. A component can contain cohesively connected classes on the condition that the component's complexity doesn't exceed a given threshold. The resulting clustered component is a more powerful functional unit. The coupling between components should eventually be low and the cohesion within a component should be high. The granularity of a component can be controlled by a complexity threshold. A component designer can help determine the limits of this metric.

#### 1. CS (Connectivity Strength)

Assume that component-based system *S'* consists of components ( $S' = \{C_1, \dots, C_k\}$ ,  $k \geq 1$ ), *CS* shows how closely connected entities including components, classes, and methods are. *CS* between methods, classes, and components can be defined

by the following formula.  $C_i$ ,  $c_v$ , and  $m_x$  represent a component, a class, and a method in the formula, respectively.

$$CS(C_i, C_j) = \sum_{c_u \in CSET(C_i)} \sum_{c_v \in CSET(C_j)} CS(c_u, c_v)$$

$$CS(c_u, c_v) = \sum_{m_x \in MSET(c_u)} \sum_{m_y \in MSET(c_v)} CS(m_x, m_y) + \sum_{m_x \in MSET(c_v)} \sum_{m_y \in MSET(c_u)} CS(m_x, m_y)$$

$$CS(m_x, m_y) = \begin{cases} |\text{pri\_arg}(m_y)| * w_{\text{pri}} + (|\text{abs\_arg}(m_y)| + 1) * w_{\text{abs}}, & \text{if } m_x \text{ calls } m_y, \\ 0, & \text{otherwise} \end{cases}$$

where

$w_{\text{pri}}$ ,  $w_{\text{abs}}$ : the weights of the type of parameter, primitive built-in type and abstract user-defined type, in order ( $w_{\text{pri}} + w_{\text{abs}} = 1.0$ , usually  $w_{\text{abs}} > w_{\text{pri}}$ ).

$|\text{pri\_arg}(m_y)|$ ,  $|\text{abs\_arg}(m_y)|$ : the number of primitive built-in type and abstract user-defined type parameters of method  $m_y$ , respectively.

## 2. Cohesion

The classes in a component must be closely related so as to provide services. To quantify their close relationship, we present a formula that measures the cohesion of a component, as follows.

$$COH(C_i) = \begin{cases} 1, & \text{if } |C_i| = 1 \\ \frac{\sum_{C_u \in CSET(C_i)} \sum_{C_v \in CSET(C_i)} w_R(c_u, c_v)}{|C_i| * (|C_i| - 1)}, & c_u \neq c_v, 1 \leq i \leq k, \text{ otherwise} \end{cases}$$

where

$$w_R(c_u, c_v) = \begin{cases} w_u, w_g, \text{ or } w_c (w_u < w_g < w_c), (c_u \neq c_v) \\ \quad \text{if } c_u \text{ has a relationship with } c_v, \\ 0, & \text{otherwise} \end{cases}$$

$w_u$ ,  $w_g$ , and  $w_c$  are a weight of use, generalization, or composition relationships, respectively.

## 3. Component Complexity

Component complexity metrics including CPC, CSC, CDC, and CCC have been suggested [9]. We derive a new component complexity metric based on CPC and CSC. Our component complexity can be defined from members of classes that belong to a component. The members of a class include inherited members from parents as well as members defined in the class. Our complexity metric focuses on structural and abstract characteristics of a component.

$$COX(C_i) = \sum_{c_u \in SET(C_i)} COX(c_u)$$

where

$$\begin{aligned} SET(C_i) &= CSET(C_i) \cup \sum_{c_j \in CSET(C_i)} GEN(c_j) \\ COX(c_i) &= \sum_{m \in MSET(c_i)} \sum_{arg \in SIGLIST(m)} \begin{cases} w_{pri}, & \text{if arg is primitive built-in type} \\ COX(arg) * w_{abs}, & \text{otherwise} \end{cases} \\ &\quad + \sum_{a \in ATTRSET(c_i)} \begin{cases} w_{pri}, & \text{if attribute a is primitive built-in type.} \\ COX(a) * w_{abs}, & \text{otherwise.} \end{cases} \end{aligned}$$

**Clustering Components.** Assuming that each class that is not included in basic components is initially a component before clustering, we group the inter-related components into a larger component. Cluster analysis is conducted on grouping components and  $CS$  and  $COH$  metrics are used as similarity measures for clustering. The more  $CS$  there is between two components, the stronger is their interconnection strength. In this study, we use a hierarchical agglomerative clustering technique. We can quantitatively measure the quality of intermediate component-based systems using the following measurement  $Q(S')$  composed of  $COH$  and  $CS$  metrics of system  $S'$ .  $CS(S')$  and  $COH(S')$  are defined as the sum of  $CS(C_i, C_j)$  and the average of  $COH(C)$  for components in system  $S'$ , respectively. The goal of clustering process is to maximize  $Q(S')$ . Two components with the highest  $CS$  among current components can be combined and an intermediate component-based system  $S'$  is produced in the clustering process. However, components whose complexity is larger than a given threshold cannot be combined because it is very difficult to understand and to reuse very complex components. The execution of the clustering process is terminated when there is no improvement in  $Q(S')$  or the complexity of every component is larger than the threshold. The best clustering is then, among the set of  $k'$  ( $k' \leq k$ ) component-based systems in the clustering process, the one that maximizes  $Q(S')$ .  $w_1$  and  $w_2$  in the following formula indicate relative importance weights on  $COH$  and  $CS$  metrics, respectively.

$$Q(S') = w_1 * \frac{COH(S')}{COH(S)} + w_2 * \frac{CS(S)}{CS(S')}$$

where

$w_1, w_2$ : importance weights on  $COH$  and  $CS$  metrics

$$CS(S') = \sum_{C_i, C_j \in S'} CS(C_i, C_j)$$

$$COH(S') = \frac{\sum_{C \in S'} COH(C)}{|S'|}$$

The clustering process is executed as follows:

1. Determine  $w_{pri}$ ,  $w_{abs}$ ,  $w_c$ ,  $w_g$ ,  $w_u$ ,  $w_j$ , and  $w_2$  weights and complexity threshold  $MAXCOX$ .
2. Initially set  $S' = S$  and compute  $Q(S')$ .
3. In any pair  $\langle C_i, C_j \rangle$  excluding the pair with which the quality function value does not improve in the previous step, find a pair  $\langle C_i, C_j \rangle$  that has the maximum of  $CS(C_i, C_j)$  only if  $COX(C_i)$  and  $COX(C_j)$  is not greater than  $MAXCOX$ . If all  $CS(C_i)$ 's ( $1 \leq i \leq k$ ) are greater than  $MAXCOX$  or all components have been examined, this process stops.
4. If  $COX(C_{i,j})$  is greater than  $MAXCOX$ , go to step 3.  $C_{i,j}$  indicates a new component that is created by merging  $C_i$  and  $C_j$ .
5. Compute  $Q(S')$  with  $COH(S')$  and  $CS(S')$ . If  $Q(S')$  is greater than  $Q(S)$ , replace  $S$  with  $S'$ . That is,  $COH(S')$ ,  $CS(S')$  and  $Q(S')$  become  $COH(S)$ ,  $CS(S)$  and  $Q(S)$ , respectively.
6. Go to step 3.

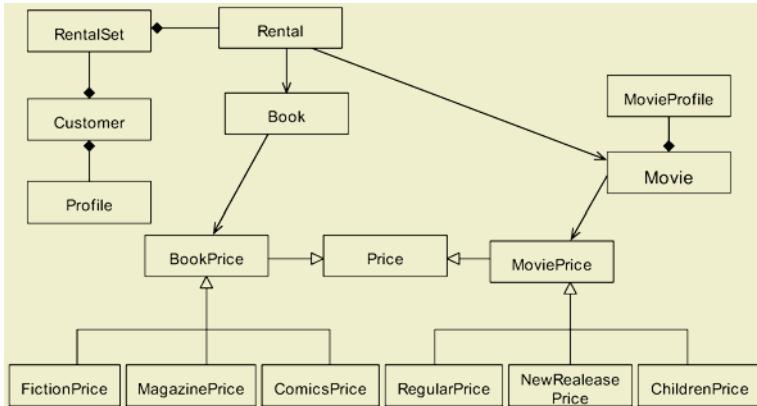
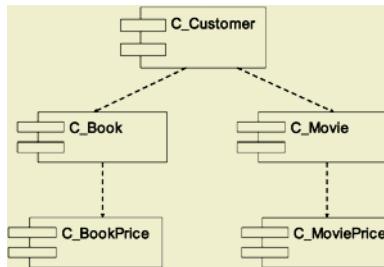
**Interface:** A component is composed of a set of classes and a set of interfaces. We have shown a process that has identified constituent classes of a component. We will now consider the creation of component interfaces. A component interface specifies services that the component provides. Among all methods in a component, it consists of only public methods that provide services to the other components or classes. Therefore, we make a component interface that includes the public methods that each class of a component has.

Component designers can help developers determine appropriate values for each weight or complexity threshold. We expect that this refining process will help create high-quality components. Moreover, by grouping related classes, we can realize a high-level view of a system using relatively large granularity components. This also allows software engineers to easily manage and reuse a system.

## 4 Applying the Process to an Example

This section describes our methodology applied to the book, movie tape rental system (Fig. 3), which was introduced in [10]. We modified the BMRS system and then implemented it in C++. It is composed of a total of 17 classes and consists of about 1500 lines of source code. The entire system excluding the ‘string’ class is described with UML diagram in Fig. 3.

In Fig. 3, a plain arrow represents a use relationship. For example, the arrow from *Rental* to *Book* in Fig. 3 indicates that class *Rental* use class *Book*. Applying the process described in Sect. 3.2 to Fig. 3 has led to Fig. 4, which shows the BMRS system composed of basic components. A dashed arrow represents a dependency relationship in Fig. 5. That is, they have the same meaning as UML notations.

**Fig. 3.** BMRS (Object-oriented).**Fig. 4.** BMRS (Basic Components).

Next, we applied our refining process to basic components in Fig. 4.  $w_{pri}$  and  $w_{abs}$  weights were set at 0.3 and 0.7, respectively, because user-defined type is usually more complex than built-in type.  $w_u$ ,  $w_g$  and  $w_c$  weights were set at 0.25, 0.35 and 0.4, respectively. A weighting scheme of 0.5 on *CS* and 0.5 on *COH* was assumed. *MAXCOX* was set at the average complexity of components in a system.

**Table 1.** CS values between basic components.

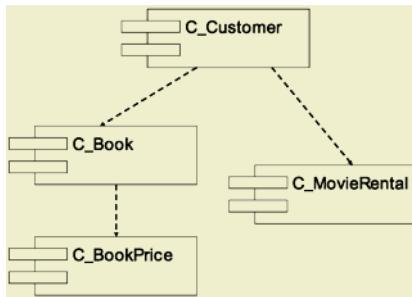
Components	<b>C_Book</b> <b>C_BookPrice</b>	<b>C_Movie</b> <b>C_MoviePrice</b>	<b>C_Customer</b> <b>C_Book</b>	<b>C_Customer</b> <b>C_Movie</b>
CS value	1.7	2.7	1.0	2.0

Table 1 shows *CS* values between basic components. The *CS* value between **C\_Movie** and **C\_MoviePrice** is larger than other *CS* values. **C\_Movie** and **C\_MoviePrice** have been merged to create a new component **C\_MovieRental** in the refining process. Fig. 5 shows the result of our refining process applied to BMRS.

**Table 2.** Measurements before and after refining components.

System	COH	CS	Q
Before refining components (Fig. 4)	0.38	7.40	1.00
After refining components (Fig. 5)	0.36	4.70	1.27

Table 2 shows the values of *COH*, *CS*, and *Q* of the systems before (Fig. 4) and after (Fig. 5) components are refined, respectively. *CS* and *Q* have improved by about 36% and about 27%, respectively, though *COH* decreased slightly.

**Fig. 5.** BMRS (Component-Based).

Developers can derive better component-based system from the result of our process using their domain knowledge and experience. For example, *COH* of component Book is defined as 1 because the component has only one class. It may be difficult to merge such a component into larger components. So developers can determine that Book and C\_BookPrice components should be merged from their knowledge and experience.

The final system is configured as follows:

```

UCOMP = {C_MovieRental, C_Customer, C_BookPrice, C_Book}
C_MovieRental = <IDC_MovieRental, I_MovieRental, {Movie,
    MovieProfile, ChildrenPrice, NewReleasePrice, Regular-
    Price, MoviePrice, Price}, φ>
C_Customer = <IDC_Customer, I_Customer, {Rental, Rental-
    Set, Customer, Profile}, {C_Book, C_MovieRental} >
C_BookPrice = <IDC_BookPrice, I_BookPrice, {ComicsPrice,
    FictionPrice, MagazinePrice, BookPrice, Price}, φ>
C_Book= <IDC_Book, I_Book, {Book}, {C_BookPrice}>
  
```

I\_Book, I\_Customer, I\_BookPrice, and I\_MovieRental are the interface of each component. We make a ‘component class’ for the interfaces to be used by other components. A component class, a kind of wrapper, is an entity class to implement a component in a legacy object-oriented language. When an interface request is received from a caller, a component class in the component (a callee) delegates the

request to an appropriate class method. Also, if necessary, it passes the results from the delegated class to the caller.

To show that this system is well executed using the interfaces, components, and other classes, we simply describe how component C\_BookPrice in C++ is used. The following code is inserted into the C\_BookPrice component.

```
#define CLASS_COMICS      "Class.ComicsPrice"
#define CLASS_FICTION     "Class.FictionPrice"
#define CLASS_MAGAZINE    "Class.MagazinePrice"
// This component-class CBookPrice is included in component C_BookPrice.
// ClassID is defined as string type and is a unique identifier among all classes.

class CBookPrice : public I_BookPrice {
private:
    BookPrice* price;
public:
    CBookPrice(ClassID id) {
        if(!strcmp(CLASS_COMICS, id)) price = new ComicsPrice;
        if(!strcmp(CLASS_FICTION, id)) price = new Fiction-
            Price;
        if(!strcmp(CLASS_MAGAZINE, id)) price = new Magazine-
            Price;
    }
    double getCharge(int daysRented) {
        return price->getCharge(daysRented);
    };
};

class I_BookPrice {
    virtual double getCharge(int daysRented) = 0;
};
```

The following shows how other classes or components can use C\_BookPrice:

```
// Using component C_BookPrice for ComicsPrice.
CBookPrice *comp = new CBookPrice(CLASS_COMICS);
IBookPrice *bp = comp; /* bp points to the interface of
    CBookPrice and used like this form,
    bp->getCharge(arg).*/
```

An instance of component class CBookPrice is created by calling a constructor method. Assuming that the ComicsPrice class is actually used, ‘CLASS\_COMICS’, which is a unique identifier of ComicsPrice, is inserted as a parameter by the constructor. I\_BookPrice, the interface of CBookPrice, can be used as an abstract class in accordance with C++ abstract classes. If there is no abstract class in a component, the ‘if’ statements, that specify the class which is actually being used, are omitted in the above code.

As shown by the previous example, our process is applicable to an object-oriented system and the steps are relatively straightforward. It is expected that the resulting component-based system will be composed of components of high quality that can be used as one functional unit.

## 5 Conclusion

In this paper, we have presented a methodology to reengineer an existing object-oriented legacy system into a component-based system. Basic components are identified based upon generalization and composition relationships between classes from program source code. Component metrics are defined to assess the quality of components and applied to refine the initial basic components. Applying clustering algorithm with the metrics in the refining process results in a component-based system with components of high quality. Since we allow only the dependency relationship between components, it is expected that each component becomes a domain-specific functional unit.

We have not addressed dynamic properties of a system including the number of method calls though addressed a call relationship between methods. Dynamic measurements of these properties may be necessary to allocate components in a distributed environment.

To date, we have developed tools that are capable of realizing a partial automation of our process. Our future work will focus on the refining process. We also hope to demonstrate the efficiency of our process by applying it to a large system using a proper heuristic algorithm.

## Acknowledgement

This work was supported by grant No.R01-1999-00238 from the Basic Research program of the Korea Science & Engineering Foundation.

## References

1. I. Sommerville. *Software Engineering* 6th ed., Addison Wesley, 2001.
2. H. Jain. Business Component Identification - A Formal Approach, *IEEE International Enterprise Distributed Object Computing Conference*, pages 183-187, 2001.
3. M. A. Serrano, C.M. de Oca, and D. L. Carver. Evolutionary Migration of Legacy Systems to an Object-Based Distributed Environment. *IEEE International Conference on Software Maintenance*, pages 86-95, 1999.
4. A. De Lucia, G. A. Di Lucca, A. R. Fasolino, P. Guerram, and S. Petruzzelli. Migrating Legacy Systems towards Object Oriented Platforms. *IEEE International Conference on Software Maintenance*, pages 122-129, 1997.
5. H. M. Sneid. 16 Generation of stateless components from procedural programs for reuse in a distributed system. *European Conference on Software Maintenance and Reengineering*, pages 183-188, 2000.
6. J. K. Lee, S. J. Jung, S. D. Kim, W. H. Jang, and D. H. Ham. Component Identification Method with Coupling and Cohesion. *Asia-Pacific Software Engineering Conference*, pages 79-86, 2001.

7. M.W. Price, D.M. Needham, and S.A. Demurjian. Producing Reusable Object-Oriented Components:A Domain-and-Organization-Specific Perspective. *Symposium on Software Reusability*, pages 41-50, 2001.
8. F. B. Abreu, G. Pereira, and P. Sousa. A Coupling-Guided Cluster Analysis Approach to Reengineer the Modularity of Object-Oriented Systems. *European Conference on Software Maintenance and Reengineering*, 2000.
9. E. S. Cho, M. S. Kim, and S. D. Kim. Component Metrics to Measure Component Quality. *Asia-Pacific Software Engineering Conference*, pages 419-426, 2001.
10. M. Fowler, *Refactoring: Improving the Design of Existing Code*, Addison Wesley, 1999.

# Communication Style Driven Connector Configurations

Tomas Bures<sup>1,2</sup> and Frantisek Plasil<sup>1,2</sup>

<sup>1</sup> Charles University, Faculty of Mathematics and Physics, Department of Software Engineering  
Malostranske namesti 25, 118 00 Prague 1, Czech Republic  
`{bures,plasil}@nanya.ms.mff.cuni.cz`  
`http://nanya.ms.mff.cuni.cz`

<sup>2</sup> Academy of Sciences of the Czech Republic, Institute of Computer Science  
`{bures,plasil}@cs.cas.cz`  
`http://www.cs.cas.cz`

**Abstract.** Connectors are used in component-based systems as first-class entities to abstract component interactions. In this paper, we propose a way to compose connectors by using fine-grained elements, each of them representing a single, well-defined function. We identify an experimentally proven set of connector elements, which, composed together, model four basic component interconnection types (procedure call, messaging, streaming, blackboard), and allow for connector variants to reflect distribution, security, fault-tolerance, etc. We also discuss how to generate such a connector semi-automatically. The presented results are based on a proof-of-the-concept implementation.

## 1 Introduction and Motivation

### 1.1 Background

“Connector” was introduced in software architectures as a first-class entity representing component interactions [21]. The basic idea is that application components contain only the application business logic, leaving the component interaction-specific tasks to connectors. However, such a characterization is too vague, since it does not strictly draw a line between component and connector responsibilities.

Different types of connectors are associated with architecture styles in [21] and analyzed as well as classified, e.g., in [11, 12]. Every architectural style is characterized by a specific pattern of components and connectors, and by specific communication styles (embodied in connectors). Thus, an architecture style requires specific connector types. For example, in the pipe-and-filter architectural style, an application is a set of filters connected by pipes. As stream is here the inherent method of data communication, the pipe connector is used to mediate a unidirectional data stream from the output port of a filter to the input port of another filter. Interestingly, the main communication styles found in software architectures correspond to the types of interaction distinguished in different kinds of middleware – remote procedure call based middleware (e.g. CORBA [16], RMI [26]), message oriented middleware (e.g. JMS [25], CORBA Message Service [16], JORAM [14]), middleware for streaming (e.g. Helix DNA [10]), and distributed shared memory (e.g. JavaSpaces [27]).

In general, a communication style represents a basic contract among components; however, such a contract has to be further elaborated when additional requirements are imposed (e.g. security, transactions). This triggers the need to capture the details not visible to components, but vital for an actual connection. This comprises the technology/middleware used to realize the connection, security issues such as encryption, quality of services, etc. These details are usually referred to as non-functional, resp. extra-functional properties (NFPs). They should be considered an important part of a connector specification, since they influence the connection behavior (reflecting these properties directly in the components' code can negatively influence the portability of the respective application across different platforms and middleware). The NFPs are addressed mainly in reflective middleware research [5,19], which actually does not consider the connectors (in terms of component-based systems), but implements a lot of their desired functionality.

To our knowledge, there are very few component models supporting connectors in an implementation, e.g. [13] and [20]. However, these component systems do not consider middleware and do not deal with NFPs. As an aside, the term “connector” can be also found in EJB [23] to perform adaptation in order to incorporate legacy systems, but capturing neither communication style nor NFPs.

## 1.2 The Goals and Structure of the Paper

As indicated in Section 1.1, a real problem with component models supporting connectors is that those existing do not benefit from the broad spectrum of functionality offered by the variety of existing middleware. Thus, a challenge is to create a connector model which would address this problem. Specifically, it should respect the choice of a particular communication style, offer a choice of NFPs, allow for automated generation of connector code, and benefit from the features offered by the middleware on the target deployment nodes. With the aim to show that this is a realistic requirement, the goal of this paper is to present an elaboration of the connector model designed in our group [3,6] which covers most of the problems above, including connector generation and removal of the middleware-related code from components.

The goal is reflected in the structure of the paper in the following way. In Section 2, we focus on the basic communication styles supported by middleware and present a generic connector model able to capture these styles and also to reflect NFPs. In the second part of Section 2 we present the way we generate connectors. In Section 3, we use the generic model to specify connector architecture for each of the communication styles with respect to a desired set of NFPs. An evaluation of our approach and related work are discussed in Section 4, while Section 5 summarizes the contributions.

## 2 Connector Model

### 2.1 Key Factors Determining a Connector

As we assume the runtime communication among components will be mediated via a middleware, the key factors determining a connector include choice of (i) communication style, (ii) NFPs, and (iii) the actual deployment of individual components.

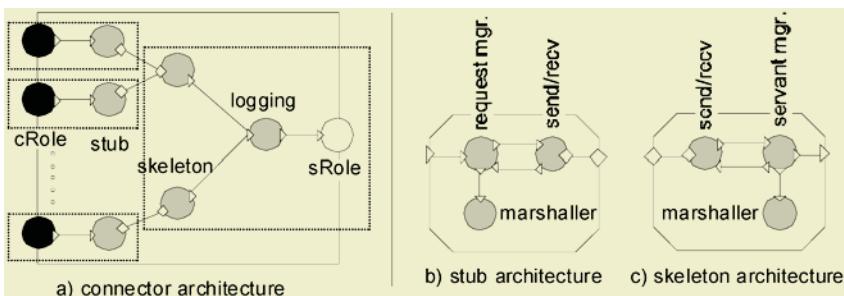
*Communication Style.* Typically, middleware supports the following communication styles: procedure call, messaging, streaming, and blackboard (Table 1). Interestingly, these interaction types correspond also to the examples of connectors in [21].

**Table 1.** Communication styles

Communication style	Description
Procedure call	Classic client server call. A client is blocked until its request is processed by a server and result is returned. <i>Example:</i> CORBA remote procedure call
Messaging	Asynchronous message delivery from a producer to subscribed listeners. <i>Example:</i> CORBA event channel service
Streaming	Uni- or bidirectional stream of data between a sender and (multiple) recipients. <i>Example:</i> Unix pipe
Blackboard	Communication via shared memory. An object is referenced by a key. Using this key the object may be repeatedly read, written, and deleted. <i>Example:</i> JavaSpaces

*NFPs.* A middleware-based implementation of a communication style can vary depending upon the desired NFPs, such as real-time constraints, middleware interoperability, monitoring, and security, as well as fault tolerance, transaction context modification, etc. In the Appendix, for each communication style, we list the NFPs we consider important and sensible in middleware-based connectors. The features are orthogonal at the same level of their hierarchy.

*Deployment.* We expect components to be deployed in a heterogenous distributed environment (with respect to the middleware available on each of the nodes composing the environment). Therefore deployment must inherently ensure interoperability of the used middleware technologies – “agreement on middleware”. In our approach, the abstraction of a node performing negotiation upon agreement on middleware, instantiation of components, and providing their run time environment is *deployment dock* [22].



**Fig. 1.** Example of a Procedure call connector

## 2.2 Architecture Level Specification

At architecture level, we model connectors using a notation based on [3], capturing connectors as a composition of connector elements which can be primitive or composed. Contrary to [3], we use composed elements to capture fine-grain parts of middleware, such as marshaling, unmarshaling, etc. Using the notation, Figure 1 shows a sample architecture of a connector reflecting the procedure call communication style, where several client components can have access to a single server component. *Roles* (circles) are in principle generic interfaces of the connector. They will be bound later to a requires (white circle) resp. provides (black circle) interface of a component. They are on the connector *frame* (its boundary, solid rectangle). Having a specific functionality, each of the *elements* provides a part of the connector implementation. In principle, the *architecture* of a connector is determined by its frame, its internal elements, and bindings between these elements' *ports* (symbols  $\triangleright$  resp.  $\diamond$ ; by convention  $\triangleright$  denotes a local and  $\diamond$  remote binding).

*Deployment units* (dotted lines) define the distribution boundaries of a connector. In principle, a deployment unit groups together the elements to be instantiated in a single address space. The responsibility for establishing a link between elements that cross unit boundaries is split between the elements on both sides of the link.

The following is an example of an architecture level connector specification in our notation:

```
/* This is the ProcedureCall connector from Figure 1
   specified in the component definition language of
   the SOFA component model (http://nenya.ms.mff.cuni.cz).
   Full version of this fragment is in [7]*/
connector frame ProcedureCall <ClientType, ServerType> {
    multiple role ClientRole {
        provides: ClientType ClientProv;
    };
    role ServerRole {
        requires: ServerType ServerReq;
    };
};

connector architecture RemoteProcedureCallImpl
implements ProcedureCall {
unit Client {
    inst EStub stub;
};
unit Server {
    inst ESkeleton skeleton;
    inst EInterceptor logging;
    bind skeleton.callOut to logging.in;
};
delegate ClientRole.ClientProv to Client.stub.callIn;
bind Client.stub.lineOut to Server.skeleton.lineIn;
bind Server.skeleton.lineOut to Client.stub.lineIn;
subsume Server.logging.out to ServerRole.ServerReq;
};
```

### 2.3 Connector Configuration

A realization of a particular connector instance is fully determined by a *connector configuration*, which is a connector architecture with implementation assigned to all of its internal elements. As the rest of this section deals with the selection of the right connector configuration, we first formalize the concept of connector configuration in order to capture all the important relationships of the related abstractions.

Let  $Arch$  be the set of available connector architectures and  $Impl$  the set of available element implementations. Combining architectures with element implementations we get a set  $Conf$  of connector configurations, where a  $c \in Conf$  determines a connector implementation. Moreover, an  $i \in Impl$  is associated with its type denoted  $Type(i)$  (e.g. ‘EInterceptor’), the function  $Elements: Arch \rightarrow P(\{<e_{name}, e_{type}>\})$  returns a set of the elements contained in  $a \in Arch$ ; here  $P(X)$  denotes the powerset of  $X$ . For the example above,  $Elements(a)$  is the set  $\{<‘skeleton’, ‘ESkeleton’>, <‘logging’, ‘EInterceptor’>, \dots\}$ . This way:

$$Conf = \{<a, s, f> \mid a \in Arch \text{ \& } s \in P(Impl) \text{ \& } f: Elements(a) \rightarrow s \text{ is a surjective mapping such that if } f(<e_{name}, e_{type}>) = i, \text{ then } Type(i) = e_{type}\}$$

Informally,  $f$  is a function that maps the elements of the architecture  $a$  to their implementations (while the type of an element and its implementation is the same).

Formalization of connectors with nested elements would be done in a similar way, except that the assignment of element implementations would be recursively applied to the nested elements.

### 2.4 Connector Construction and Instantiation

A connector is constructed and instantiated in the following three steps:

A connector configuration  $<a, s, f>$  is selected with respect to the desired communication style, NFPs, and deployment.

1. The generic connector roles are anchored to the specific component interfaces causing the generic interfaces in the elements in  $s$  to be anchored as well (element adaptation).
2. The adapted element implementations are instantiated at runtime and bound together according to  $a$ .

Described in [6], the steps 2 and 3 are rather technical. The step 1 is more challenging since it involves both choosing the “right” architecture  $a$  and finding the “right” elements fitting into  $a$ . In our approach we traverse the whole set  $Conf$  looking for such a configuration that would comply with the desired communication style, NFPs, and deployment. Given a configuration  $c \in Conf$ , these three factors are reflected in the following consistency rules:

- a. *Communication style consistency*. The configuration  $c$  implements the desired communication style.
- b. *NFP consistency*. The configuration  $c$  complies with all desired NFPs.

- c. *Environment consistency.* All element implementations used in the configuration  $c$  are compatible with the target platform (all necessary libraries are present, etc.).

In general, the set  $Conf$  may contain configurations in which the elements cannot inherently cooperate (e.g., a configuration for the procedure call communication style featuring a CORBA-based stub, but RMI-based skeletons). To identify such configurations, we introduce:

- d. *Cooperation consistency.* The element implementations in  $c$  can cooperate (they can agree on a common middleware protocol, encryption protocol, etc.).

## 2.5 Choosing the Right Configuration

The four consistency rules above allow to automate the choice of an appropriate connector configuration by employing constraint programming techniques (e.g. backtracking). To employ the consistency rules in such a technique, we formulate the rules as logical predicates and illustrate their construction bellow.

*NFP Consistency:* It checks whether, for a particular connector configuration  $c \in Conf$ , a NFP  $p$  (e.g. *distribution*) has a specific value  $v$  (e.g. ‘local’, ‘CORBA’, ‘RMI’, ‘SunRPC’).

First of all, we associate every element implementation  $i \in Impl$  with a predicate to check whether the value of an arbitrary NFP  $p$  is  $v$ . For example, considering an RMI implementation of a stub element, such predicate can take the following form (in Prolog notation):

```
nfp_mapping_distrib_RMISubImpl('distribution', 'RMI',
    ConConfig) :- con_arch_name(ConConfig, 'RMISubImpl').
```

Now, to check the same NFP at the architecture level, we associate every architecture  $a \in Arch$  with a “higher-level” predicate referring to all predicates that check  $p$  at element level. Technically, we write a generic predicate which will be employed whenever a configuration with architecture  $a$  is used (the predicate takes configuration as a parameter). For example, for the architecture from Figure 1a such predicate would be:

```
nfp_mapping_distrib_RPCImpl('distribution', NFPValue,
    ConConfig) :- 
    (con_arch_name(ConConfig, 'RemoteProcCallImpl'),
    con_elem(ConConfig, 'Stub'),
    nfp_mapping('distribution', NFPValue, StubElemConfig)).
```

Finally, to get a single predicate checking an arbitrary NFP of any configuration, we compose all the previously defined predicates (for architectures and elements) using disjunction:

```
nfp_mapping(NFPName,NFPVal,ConConfig) :-
    nfp_mapping_distrib_RPCImpl(NFPName,NFPVal,ConConfig) ;
    nfp_mapping_distrib_RMISubImpl(NFPName,NFPVal,ConConfig) ;
    ...
.
```

The whole trick is that a term in the disjunction chain fails if it cannot “say anything about a given configuration”.

*Communication Style Consistency:* Each of the communication styles we consider (Table 1) is reflected in a very specific connector frame (there is no frame reflecting two communication styles). Therefore, a communication style consistency check results in a simple test whether a particular architecture really implements the frame corresponding to the desired communication style.

*Environment Consistency:* In order to identify the element implementations which would not run properly on a target platform, we associate each element implementation with a set of environment requirements that have to be met by the target platform. For the time being, we use a simple set of key-value pairs (such as “SunRCP”=>“2”) for the requirement specification. Every target deployment node declares its “environment provisions” in form of key-value pairs as well. Environment consistency is then guaranteed if, for each element implementation used in a configuration, its set of environment requirements is a subset of the environment provisions of the deployment node where the element is to be instantiated. This approach is very simplistic though, not allowing to express complex requirements. (We plan to design a more sophisticated requirement specification technique in the future.)

*Cooperation Consistency:* Cooperation consistency guarantees that all the element implementations in a configuration are able to cooperate (i.e. “they speak the same protocol”). Cooperation consistency is verified in two steps::

1. Every element implementation  $i \in Impl$  is associated with a *transition predicate*, which expresses a relation of interfaces among the ports of  $i$ . This describes “how an interface changes” when mediated by a particular element implementation.
2. For every binding in the architecture of a given configuration, we construct a *binding predicate*, which captures the fact that two bound elements must use subtype compatible [18] interfaces to communicate with each other.

Finally, all the *transition predicates* and *binding predicates* are tied together using conjunction to form a composed predicate able to check cooperation consistency for a given configuration.

### 3 Building Real Connectors

By analyzing several middleware designs and implementations [16, 26, 25, 14, 10, 27], we have identified a list of NFPs which can be addressed in middleware (see Appendix). In this section, we suggest a connector architecture for Procedure call and

Messaging communication styles reflecting a relevant spectrum of the identified NFPs. The other two communication styles (Streaming and Blackboard) are omitted due to space constraints. Their detailed description can be found in [7]. Similar to the example from Section 2, we reflect a single NFP in one or more connector elements organized in a specific pattern to achieve the desired connector functionality.

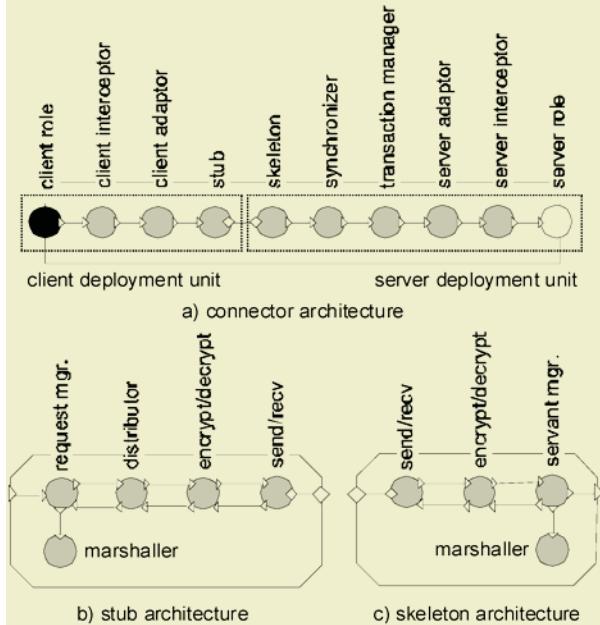


Fig. 2. Proposed procedure call connector architecture

### 3.1 Procedure Call

The proposed connector architecture for the procedure call communication style is depicted in Figure 2a. It consists of a server deployment unit and multiple client deployment units. For simplicity, only one client deployment unit is shown. The other client units (identical in principle) are connected to the server deployment unit in the way illustrated in Figure 1.

In summary, the connector NFPs (listed in Appendix) are mapped to elements as described below. In principle, distribution is mapped to the existence of stubs and skeletons, and the NFPs dependent on distribution are reflected by the existence or variants of the elements inside the stub or the skeleton – encryption by an encrypt/decrypt element, connection quality by a send/recv element, and fault-tolerance by a distributor element replicating calls to multiple server units.

In more detail, the functionality of particular elements is following:

- *Roles.* Not reflecting any NFP, roles form the connector entry/exit generic points.
- *Interceptors* reflect the *monitoring property*. In principle, they do not modify the calls they intercept. If monitoring is not desired, these elements can be omitted.

- *Adaptors* implement the *adaptation property*. They solve minor incompatibilities in the interconnected components' interfaces by modifying the mediated calls. An adaptation can take place on the client side as well as on the server side (thus affecting all clients). If no adaptation is necessary, the elements can be omitted.
- *Stub*. Together with a skeleton element, the stub implements the *distribution property*. This element transfers a call to the server side and waits for a response. The element can be either primitive (i.e. directly mapped to the underlying middleware) or compound. A typical architecture of a stub is in Figure 2b. It consists of a request manager, which, using the attached marshaller, creates a request from the incoming call and blocks the client thread until a response is received. An encryption element reflects the *encryption property*; sender/receiver elements transport a stream of data and also reflect the *connection quality property*. The *fault-tolerance property* is implemented by a distributor performing call replication. The stub element is needed only when distribution is required.
- *Skeleton* is the counterpart of the stub element. Again, its architecture can be primitive or compound (Figure 2c). The elements in the compound architecture are similar to those in the compound stub. The servant manager uses the attached marshaller to create a call from the received data and assigns it to a worker thread. Again, skeleton can be omitted if distribution is not required.
- *Synchronizer* reflects the *threading policy property*. It synchronizes the calls going to the server component, allowing, e.g., a thread-unaware code to work properly in a multithreaded environment.
- *Transaction mgr.* implements the *transaction property*. When appropriate, it can modify the transaction context of the mediated calls.

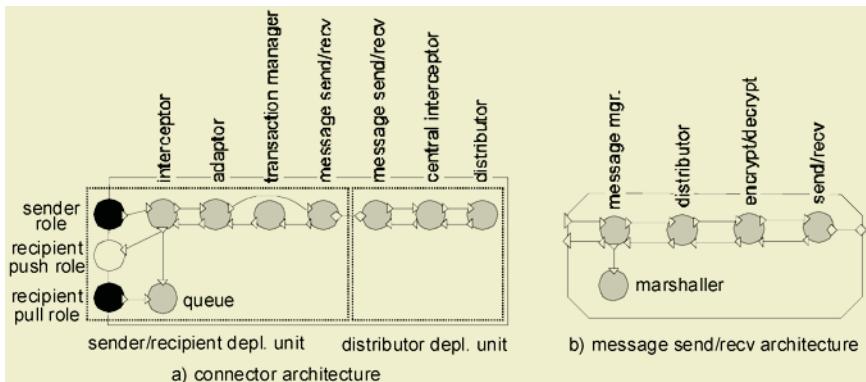
### 3.2 Messaging

The proposed connector architecture for the messaging communication style is depicted in Figure 3. It consists of a distributor deployment unit and several sender/recipient units. (In a fault-tolerant case, there can be multiple distributor deployment units.) The sender/recipient deployment unit allows sending messages to attached components (as well as for receiving messages from them). The distributor deployment unit is in the middle of this logical star topology. For simplicity, only one sender/recipient deployment unit is shown. Other sender/recipient units would be connected to the distributor deployment unit in a similar way.

The connector NFPs (listed in Appendix) are mapped to elements as described below.

- *Roles*. The sender role serves for sending messages. Depending on the *recipient mode property*, the reception can work either in push mode, employing the push role to automatically deliver the incoming messages to the attached component via a callback interface; or in pull mode, when the attached component polls for new messages via the pull role. If the component does not need to receive messages, the recipient role can remain unconnected.

- *Queue*. Together with the pull role, this implements the pull variant of the *recipient mode property*. Thus, the queue is present only when the message reception works in pull mode to buffer the incoming messages if necessary.
- *Interceptors* implement the *monitoring property* (similarly to the procedure call architecture).
- *Adaptor* reflects the *adaptation property* by modifying the mediated messages.
- *Transaction mgr.* implements the *transaction property*. Its presence is meaningful only if message reception operates in push mode.
- *Message sender/receiver* realize the *distribution property*. Each of them performs communication with remote nodes. They can be either primitive (directly implemented by underlying middleware) or compound (its typical architecture is on Figure 3b). It is similar to the stub element, however the request manager is replaced by a message manager which allows the messages to be transferred in both directions. The distributor deployment unit supports implementation of the *fault-tolerance property*.
- *Distributor*. Being inherent to the communication style, it is a central part of the connector architecture. It distributes all the incoming messages to the attached recipient components. The element reflects the *delivery strategy property* by implementing different policies for message routing (one recipient, all recipients, group address, etc.).



**Fig. 3.** Proposed messaging connector architecture

## 4 Evaluation and Related Work

To our knowledge, there is no related work addressing all of the following issues in a single component model and/or in its implementation: 1) Reflecting the component interaction types which are supported by existing middleware, 2) providing the option of choosing from a set of NFPs, and 3) generation of a connector with respect to the middleware available on target deployment nodes.

1. *Component Interactions.* We have identified four basic communication styles that are directly supported by middleware (i.e. procedure call, messaging, streaming, blackboard). These styles correspond to the connector types mentioned in software architectures in [21]. Medvidovic et al. in [12] go further and propose additional connector types (adaptor, linkage, etc.); in our view, being at a lower abstraction level, these extra connector types are potential functional parts of the four basic connector types (e.g., adaptation may be associated with any of the communication styles in our approach).
2. *NFPs.* We have chosen an approach of reflecting a specific NFP as a set of reusable connector elements. Following the idea of capturing all the communication-related functionality in a connector (leaving the component code free of any middleware dependent fragments), we have managed to compose the key connector types in such a way that NFPs are realized via connector elements and a change to a NFP implies only a replacement of few connector elements, leaving the component code untouched. Our approach is similar to reflective middleware [5, 4, 9, 19], which is also composed of smaller parts; however, middleware-dependent code is inherently present in components, making them less portable. Our work is also related to [8] and [2]. The former proposes a way to unify the view on NFPs influencing quality of service in real-time CORBA. It does not consider different communication styles, connectors as communication mediators, and relies on having the source code of both the application and the middleware available. The latter describes how to incorporate connectors into the ArchJava language [1]. It allows to reflect NFPs in connectors too, but at the cost that the whole connector code has to be completely coded by hand.
3. *Automatic Generation.* We have automated the process of element selection and adaptation, and connector instantiation; however we plan to automate, to a certain degree, the design process of a connector architecture. The idea of generating middleware-related code in an automatic way is employed in ProActive [15], where stubs and skeletons are generated at run-time. However, ProActive is bound only to Java not considering other communication styles than RPC and not addressing NFPs.

*Prototype Implementation:* As a proof of the concept, a prototype implementation of a connector generator for the SOFA component model [17] is available, implementing three of the four considered communication styles (procedure call, messaging, and datastream) [22]. Designed as an open system employing plugins for easy modification it consists of two parts. The first one, written in Prolog, takes care of selecting a particular architecture and element implementations with respect to given communication style and NFPs (as described in Section 2) and produces a connector configuration. The second part, written in Java, takes the produced connector configuration and adapts its element implementations to the actual interfaces and, later on, instantiates a connector.

## 5 Conclusion and Future Intentions

In this paper, we presented a way to model and generate “real connectors” employing existing middleware. We have elaborated the connector model initially proposed in [3] to reflect the commonly used communication styles, as well as non- and extra-

functional properties. In addition to separating the communication-related code from the functional code of the components, the model allowed us to partially generate connectors automatically to respect (i) the middleware available on the target nodes determined by component deployment, and (ii) the desired communication style and NFPs. Our future intentions include addressing the open issue of finding a metric allowing to select the “best” configuration when more than one configuration, meeting the desired factors, is found.

## Acknowledgments

We would like to give special credit to Lubomir Bulej, a coauthor of [6]. Special thanks go to Petr Tuma, Jiri Adamek and other colleagues in our group for their valuable comments. Also, Petr Hnetyntka deserves special credit for incorporating the implementation of connector generator into the SOFA framework. This work was partially supported by the Grant Agency of the Czech Republic (project number 201/03/0911); the results will be employed in the OSMOSE/ITEA project.

## References

1. J. Aldrich, C. Chambers, D. Notkin, “ArchJava: Connecting Software Architecture to Implementation”, in Proceedings of ICSE 2002, May 2002
2. J. Aldrich, V. Sazawal, D. Notkin, C. Chambers, “Language Support for Connector Abstractions”, in Proceedings of ECOOP 2003, Darmstadt, Germany, July 2003
3. D. Balek, F. Plasil, “Software Connectors and Their Role in Component Deployment”, in Proceedings of DAIS'01, Krakow, Kluwer, Sept. 2001
4. G. Blair, et al., “The Role of Software Architecture in Constraining Adaptation in Component-based Middleware Platforms”, in Proceedings of Middleware 2000, IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, Hudson River Valley (NY), USA. Springer Verlag, LNCS, April 2000
5. G. Blair, et al., “A Principled Approach to Supporting Adaptation in Distributed Mobile Environments”, International Symposium on Software Engineering for Parallel and Distributed Systems, Limerick, Ireland, June 2000
6. L. Bulej, T. Bures, “A Connector Model Suitable for Automatic Generation of Connectors”, Tech. Report No. 2003/1, Dep. of SW Engineering, Charles University, Prague, 2003
7. T. Bures, F. Plasil, “Composing connectors of elements”, Tech. Report No. 2003/3, Dep. of SW Engineering, Charles University, Prague, 2003
8. J. K. Cross, D. C. Schmidt, “Quality Connectors. Meta-Programming Techniques for Distributed Real-time and Embedded Systems”, the 7<sup>th</sup> IEEE Workshop on Object-oriented Real-time Dependable Systems, San Diego, January 2000
9. B. Dumant, F. Horn, F. Dang Tran, J.-B. Stefani, “Jonathan: an Open Distributed Processing Environment in Java”, 1998
10. Helix Community, “Helix DNA”, [www.helixcommunity.org](http://www.helixcommunity.org)
11. N. Medvidovic, N. R. Mehta, “Distilling Software Architecture Primitives from Architectural Styles”, TR UCSCSE 2002-509
12. N. Medvidovic, N. R. Mehta, S. Phadke, “Towards a Taxonomy of Software Connectors”, in Proceedings of the International Conference on Software Engineering, Limerick, Ireland, June 2000
13. N. Medvidovic, P. Oreizy, R. N. Taylor, “Reuse of Off-the-Shelf Components in C2-Style Architectures”, in Proceedings of the 1997 International Conference on Software Engineering (ICSE'97), Boston, MA, 1997

14. ObjectWeb Consortium, "JORAM: Java Open Reliable Asynchronous Messaging", [www.objectweb.org/joram](http://www.objectweb.org/joram)
15. ObjectWeb Consortium, "ProActive manual version 1.0.1", January 2003
16. OMG formal/02-12-06, "The Common Object Request Broker Architecture: Core Specification, v3.0", December 2002
17. F. Plasil, D. Balek, R. Janecek, "SOFA/DCUP: Architecture for Component Trading and Dynamic Updating", in Proceedings of ICCDS'98, Annapolis, Maryland, USA, IEEE CS Press, May 1998
18. F. Plasil, S. Visnovsky: Behavior Protocols for Software Components, IEEE Transactions on Software Engineering, vol. 28, no. 11, Nov. 2002
19. J. Putman, D. Hybertson, "Interaction Framework for Interoperability and Behavioral Analyses", ECOOP Workshop on Object Interoperability, 2000
20. M. Shaw, R. DeLine, G. Zalesnik, "Abstractions and Implementations for Architectural Connections", in Proceedings of the 3rd International Conference on Configurable Distributed Systems, May 1996
21. M. Shaw, D. Garlan, Software Architecture, Prentice Hall, 1996
22. The SOFA Project, <http://sofa.debian-sf.objectweb.org/>
23. Sun Microsystems, Inc., "Enterprise JavaBeans Specification 2.0, Final Release", August 2001
24. Sun Microsystems, Inc., "Java IDL", <http://java.sun.com/j2se/1.4.1/docs/guide/idl/index.html>
25. Sun Microsystems, Inc., "Java Message Service", April 2002
26. Sun Microsystems, Inc., "Java Remote Method Invocation Specification – Java 2 SDK, v1.4.1", 2002
27. Sun Microsystems, Inc., "JavaSpaces Service Specification", April 2002

## Appendix

Procedure Call	
Feature name	Comment
distribution	The connection may be either in one address space or span across several address spaces and/or computer nodes.
Distributed	encryption
	Encryption can be employed to provide security even on insecure lines.
	connection quality
fault-tolerance	It may be necessary to assure some quality of connection (e.g. maximal latency, throughput etc.)
	The connector can support replication to make the server fault-tolerant.
threading policy	The calls may be serialized (single-threaded) or left unchanged.
adaptation	Both the calls and their parameters may be modified in order to allow incompatible component interfaces to cooperate.
monitoring	The calls and their parameters may be monitored to allow for profiling and other statistics (usage, throughput, etc.)
transactions	This feature specifies how to handle the transactional context (e.g. propagate the clients' transaction at the callee side).

Streaming		
	Feature name	Comment
	distribution	The data may be exchanged within only one address space or across several address spaces and computers.
Distributed	encryption	Encryption can be employed to provide security even on insecure lines.
	connection quality	It may be necessary to assure some quality of connection (e.g. maximal latency, throughput, etc.).
	fault-tolerance	Allows for groups of replicas instead of single recipients making the application fault tolerant.
	adaptation	The transmitted stream may be modified in order to allow incompatible components to cooperate.
	monitoring	The transmitted messages may be monitored allowing for profiling and other statistics (usage, throughput, etc.).
	duplexity	The connector may be either unidirectional (half-duplex) or bidirectional (full-duplex).
half-dup.	multicast	If the connector is half-duplex, the stream can have more recipients, allowing for e.g. audio and video broadcasting.
	recipient pull/push mode	Every recipient can work either in pull or push mode. In push mode the received data are immediately given to recipient (the recipient “receive” method is invoked). In pull mode the recipient actively polls for incoming data.

Messaging		
	Feature name	Comment
	distribution	The messages may be exchanged within only one address space or across several address spaces and computers.
Distributed	encryption	Encryption can be employed to provide security even on insecure lines.
	connection quality	It may be necessary to assure some quality of connection (e.g. maximal latency, etc.).
	fault-tolerance	Allows to groups of replicas instead of single recipient making the application fault tolerant.
	adaptation	The transmitted messages may be modified in order to allow incompatible components to cooperate.
	monitoring	The transmitted messages may be monitored allowing for profiling and other statistics (usage, throughput, etc.).
	transactions	This feature specifies how to handle the transactional context (e.g. requires, requires new, etc.).
	delivery strategy	This feature controls to whom the message should be delivered. Possible values may be: exactly one, at least one, all.
	recipient pull/push mode	Every recipient can work either in pull or push mode. In push mode every new message is immediately given to recipient (the recipient “accept message” method is invoked). In pull mode the recipient actively polls the incoming queue for new messages.

<b>Blackboard</b>		
Feature name		Comment
distribution		The data may be shared by components residing only in one address space or by components spanned across networks.
Distributed	encryption	Encryption can be employed to provide security even on insecure lines.
	connection quality	It may be necessary to assure some quality of connection (e.g. maximal latency, throughput, etc.).
adaptation		The accessed values may be transparently modified in order to allow incompatible components to cooperate.
monitoring		The accessed data may be monitored allowing for profiling and other statistics (usage, throughput, etc.).
locking		An attached component may obtain a lock onto a set of keys. The other components accessing the same data are temporarily blocked.

# A Study on the Specification for e-Business Agent Oriented Component Based Development

Haeng-Kon Kim<sup>1</sup>, Hae-Sool Yang<sup>2</sup>, and Roger Y. Lee<sup>3</sup>

<sup>1</sup> Department of Computer Information & Communication Engineering  
Catholic University of Daegu  
Kyungsan, Kyungbuk 712-702, South Korea  
hangkon@cu.ac.kr

<sup>2</sup> Graduate School of Venture, HoSeo Univ., Bae-Bang myon, A-San  
Chung-Nam 336-795, South Korea  
hsyang@office.hoseo.ac.kr

<sup>3</sup> Dept. of Computer Science, Central Michigan Univ., Mt.Pleasant, MI 48859, USA  
lee@cps.cmich.edu

**Abstract.** Agent technology becomes more and more importance in the e-business domain. The concepts and technology have been brought to a stage where they are useable in real applications, and there is a growing understanding of how to apply them to practical problems. Component methodologies have proved to be successful in increasing speed to market of software development projects, lowering the development cost and providing better quality. In this paper, we propose systemical development process using component and UML(Unified Modeling Language) technology to analysis, design and develop e-business agent. The ebA-CBD(e-business Agent-Component Based Development) process is an attempt to consider all of the best features of existing AOSE(Agent Oriented Software Engineering) methodologies while grounding agent-oriented concepts in the same underlying semantic framework used by UML, the standard Modeling language for Object Oriented Software Engineering. Finally we describe how these concepts may assist in increasing the efficiency and reusability in business application and e-business agent development.

**Keywords:** E-Business Agent(ebA), ebA-CBD Reference Architecture, ebA-Spec., Component Based Development

## 1 Introduction

Recently the software lifecycle is getting shorter and web service for paradigm of next generation information technology is more focused on while e-business model has developed very rapidly. Therefore, the development of software which is more functionable, various, stable software, the key of business domain. According to these requirements, not only the component having exchangeable module that performs independent business and function in software system but also the utilization of agent in e-business domain become more noticeable. It is important to produce the agent

service based on component technology that is change to replacement and portability toward developing the software having high productability [1][2].

In this paper, we propose a process and modeling method to apply ebA-CBD with role model, goal model, architecture model and communication model in the view of agent. We identify the 4 different models considered as ebA model. The ebA model can define agent characteristics and the relations among agents. The related component can be developed with ebA specification.. In addition we apply the process and model into component information search agent as a case study.

## 2 Related Works

### 2.1 Agent Concept Model

An agent is an atomic autonomous entity that is capable of performing some useful function. The functional capability is captured as the agent's services. A service is the knowledge level analogue of an object's operation. The quality of autonomy means that an agent's actions are not solely dictated by external events or interactions, but also by its own motivation. We capture this motivation in an attribute named purpose. The purpose will, for example, influence whether an agent agrees to a request to perform a service and also the way it provides the service. Software Agent and Human Agent are specialization of agent[3].

Figure 1 gives an informal agent-centric overview of how these concepts are inter-related. The role concept allows the part played by an agent to be separated logically from the identity of the agent itself. The distinction between role and agent is analogous to that between interface and class: a role describes the external characteristics of an agent in a particular context. An agent may be capable of playing several roles, and multiple agents may be able to play the same role. Roles can also be used as indirect references to agents. This is useful in defining re-usable patterns. Resource is used to represent non-autonomous entities such as databases or external programs used by agents. Standard object-oriented concepts are adequate for modeling resources[4].

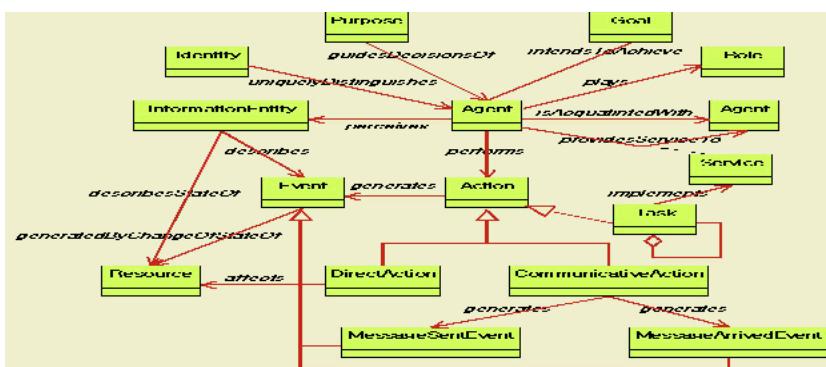


Fig. 1. Agent Concept Model.

## 2.2 e-Business Dynamic Agents

To approach E-business automation, agents need to have dynamic behavior while maintaining its identity and consistent communication channels, as well as retaining data, knowledge and other system resources to be used across applications.

It is neither sufficient for an agent to be equipped with a fixed set of application-specific functions, nor practical for the above capabilities to be developed from scratch for each agent. This has motivated us to develop a *dynamic-agent* infrastructure [4,5]. The infrastructure is Java-coded, platform-neutral, lightweight, and extensible. Its unique feature is the support of *dynamic behavior modification* of agents; and this capability differentiates it from other agent platforms and client/serverbased infrastructures.

A dynamic-agent has a *fixed part* and a *changeable part*. (Figure 2) As its fixed part, a dynamic-agent is provided with light-weight, built-in management facilities for distributed communication, object storage and resource management. A dynamic agent is capable of carrying data, knowledge and **programs** as objects, and executing the programs. The data and programs carried by a dynamic agent form its changeable part. All newly created agents are the same; their applicationspecific behaviors are gained and modified by dynamically loading Java classes representing data, knowledge and application programs. Thus dynamic-agents are general-purpose **carriers** of programs, rather than individual and applicationspecific programs. The architecture of dynamic-agent can be explained in more detail by the following.

**Built-in Facilities.** Each dynamic-agent is provided with several light-weight built-in facilities for managing messaging, data and program object storage, action activation, GUI, etc. A carried application, when started, is offered a reference to the underlying built-in management facilities, and can use this reference to access the APIs of the facilities. A *message-handler* is used to handle message queues, sending, receiving and interpreting inter-agent messages. The interaction styles include one-way, request/reply, and publish/subscribe(selective broadcast). Message forwarding is also supported. An *action-handler* is used to handle the message-enabled instantiation and execution of application programs (Java classes). One dynamic-agent can carry multiple action programs. An *open-server-handler* is used to provide a variety of continuous services, which can be started and stopped flexibly at dynamic-agent runtime. A *resource-handler* is used to maintain an object-store for the dynamic-agent, that contains application-specific data, Java classes and instances including language interpreters, addresses and any-objects (namely, instances of any class). Applications executed within a dynamic-agent use the built-in dynamic-agent management facilities to access and update application-specific data in the object-store, and to perform inter-agent communication through messaging.

**Dynamic Behavior.** Enabled by corresponding messages, a dynamic-agent can load and store programs as Java classes or object instances, and can instantiate and execute the carried programs. Within these programs, built-in functions can be invoked to access the dynamic-agent's resources, activate and communicate with other actions run on the same dynamic agent, as well as communicate with other dynamic-agents or even stand-alone programs.

**Mobility.** Two levels of mobility are supported. A dynamic agent may be moved to or cloned at a remote site. Programs carried by one dynamic-agent may be sent to another, to be run at the receiving site.

**Coordination.** Every dynamic-agent is uniquely identified by its symbolic name. Similar to FIPA [13], a coordinator agent is used to provide naming service, mapping the name of each agent to its current socket address. The coordinator is a dynamic-agent with the added distinction that it maintains the agent name registry and, optionally, resource lists. When a dynamic-agent, say  $A$ , is created, it will first attempt to register its symbolic name and address with the coordinator by sending a message to the coordinator. Thereafter,  $A$  can communicate with other dynamic-agents by name. When  $A$  sends a message to another whose address is unknown to  $A$ , it consults the coordinator to obtain the address. If  $A$  is instructed to load a program but the address is not given, it consults the coordinator or the request sender to obtain the address. Each dynamic agent also keeps an address-book, recording the addresses of those dynamic agents that have become known to it, and are known to be alive.

In a multi-agent system, besides naming service, other coordination may be required, and provided either by the coordinator or by other designated dynamic-agents that provide brokering services. A **resource-broker** maintains a hierarchically structured agent capability registry. The leaf-level nodes referring to the dynamic agents carry corresponding programming objects (action modules). Agents contact the resource-broker when acquiring or dropping new programming objects, and when they need a program such as a domain specific XML interpreter. A **request-broker** is used to isolate the service requesters from the service providers (i.e. dynamic agents that carry the services) allowing an application to transparently make requests for a service. An **event-broker** delivers events, treated as asynchronous agent messages, to event subscribers from event generators. Event notification may be point-to-point, where the event subscribers know the event generators and make the subscriptions accordingly; or multicast, where one or more event-brokers are used to handle events generated and subscribed anywhere in the given application domain. These event distribution mechanisms allow agents to subscribe to events without prior knowledge of their generators. Dynamic-agents may form hierarchical groups, with a coordinator for each group. Based on the group hierarchy a multilevel name service can be built.

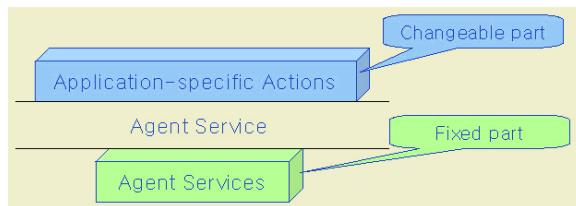


Fig. 2. e –Business Dynamic Agent.

**Dynamic Role Assignment and Switching.** A role is the abstraction of one or more agents, providing a dynamic interface between agents and cooperative processes.

Agents playing a specific role follow the normative rules given by the cooperative process. The action carrying capability, together with other capabilities of dynamic agents, make the mapping from a role to a specific agent extremely simple, and allows one agent to play different roles, even simultaneously. In fact, a dynamic agent obtains the capability for playing a role simply by downloading the corresponding programs, passes a role to another agent by uploading the programs, or switches roles by executing different programs it carries. In order for an agent to switch its roles back and forth, the agent must be equipped with memory capability across multiple activities, and possess a consistent communication channel. For example, an agent responsible for ordering a product may well be the one responsible for order cancellation. Such history sensitivity and identity consistency are exactly the strength of our dynamic agents. For example, adjusting load balance by task reallocation is a kind of agent cooperation. Reallocation is beneficial when tasks are not initially allocated to the agents that handle them least expensively. Each agent can contract out tasks it had previously contracted in, give out if the task is more suitable for another agent to perform, or accept a task if the task is more suitable to be done by itself than by another. The action carrying and exchanging capability of dynamic agents naturally support task reallocation. Dynamic agents form a dynamic distributed computing environment for E-Commerce. In the following sections we shall show our solutions to E-Commerce automation that take advantage of this.

### 3 ebA-CBD Process

As we suggested ebA-CBD reference architecture in previous research[7], component development process based architecture is a set of activities and associated results, which lead to the production of a component as shown in figure 4. These may involve the development of component from ebA specification by using UML model. In addition, we consider systematical development process using AUML and ebA model technology to analyze, design, and develop e-business agent. The domain analysis specification, design model, implemented component, which are produced though the process, are stored in the repository. We use a CRMS(Component Repository Management System).

#### 3.1 Reference Architecture of ebA-CBD

In order to construct component reference architecture, agent is classified in general agent type and e-business function attribute. Figure 3 is a component and meta architecture of based on all above described for e-business agent[6][7].

Reference architecture is consisted of dimension, which has 14 general types and 11 concrete business agent types with domain oriented component architecture. These two classification areas tend to be independent for each cross-referenced. Each area has its own horizontal and vertical characteristics. General agent types are corresponding to agent platform and application. It is possible to develop agent system or application by the referencing architecture. The technology of agent can be applied to business domain.

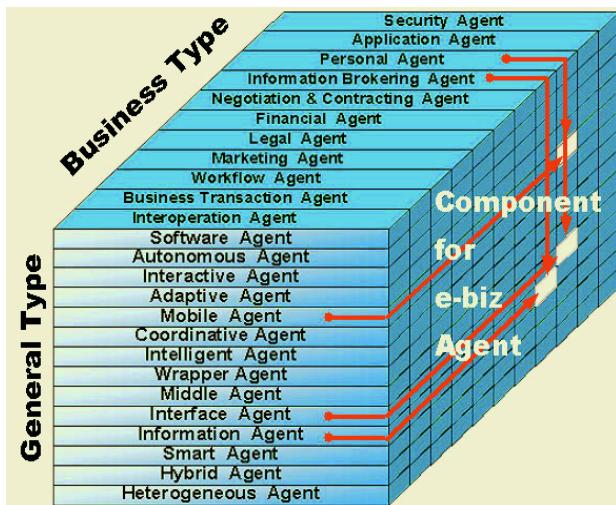


Fig. 3. ebA-CBD Reference Architecture.

A developed component is classified by the reference architecture and is placed according to general agent type and business attribute. In case agent is applied to the agent system or business domain, system is possibly to build up by identifying component related to business domain and combining it.

### 3.2 ebA Requirements Identification Phase

The requirement of agent should be first identified in desired business system. The primary property of agent is able to analyze after that the description for specific agent platform and the sorts of essential properties should be understood. At the same time, it is very important to consider weather the requirement, which is already defined, is corresponding to agent type in reference architecture and what business concept is focused on..

For the problem domain analysis, UML approach is used. With UML, not only static but dynamic aspects of system can be analyzed and agent elicitation can be conducted. Diagrams used in problem domain analysis are represented below.

1. *UseCase Diagram*: A UseCase diagram is a diagram that shows a set of use cases and actors and their relationships. It supports the behavior of a system by modeling static aspects of a system.
2. *Sequence Diagram*: Sequence diagram shows an interaction, consisting of a set of objects and their relationships, emphasizing the time ordering of messages. It models dynamic aspects of a system.
3. *Class Diagram*: Class diagram shows a set of classes, interfaces, and collaborations and their relationships. It address the static design view of a system, showing a collection of declarative elements.

4. *Activity Diagram:* Activity diagram shows the flow from activity to activity. Activities ultimately result in some action, which is made up of executable atomic computations that result in a change in state of the system or the return of a value. It is used to provide dynamic aspects of a system such as system functions.

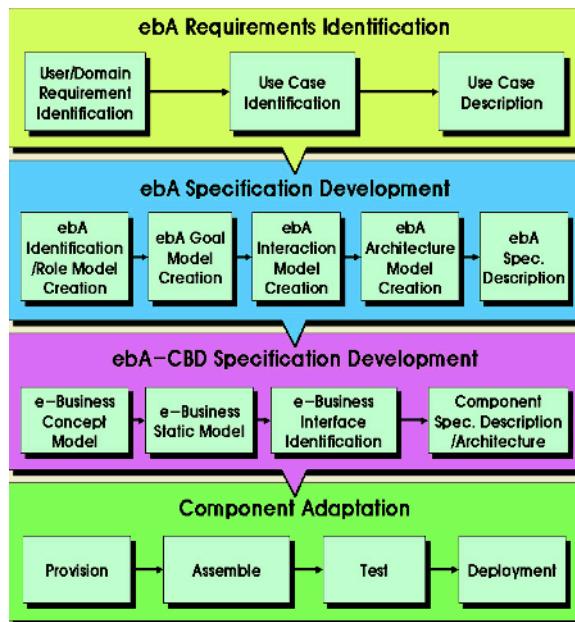


Fig. 4. ebA-CBD process.

### 3.3 ebA Specification Development

Agent specification based on user's requirement creates ebA Specification and 4 different models. These products of the specification phase becomes the main resource to identify and development new components.

ebA Model extends the UML meta-model by adding a number of elements to provide powerful expressiveness. In this section, we describes the notation that ebA Model for representing graphical instances of new meta-elements in the diagrams.

Figures 5 provide a summary of the symbols representing the ebA Model concepts and relations respectively. It is notice that symbols are associated to elements natively included in the UML meta-model. Relational diagrams are used to represent the dynamic aspects of a component. A relational diagram describes the states a agent component can be in during its life cycle along with the behavior of the component while in those states

The usages of relationships are as follows:

- Implication: This relation links one or more elements that have an attribute of type state to a single element that has an attribute of type state.

- Assignment: This relation links an element of type Autonomous Entity to an element that has an attribute of type Autonomous Entity. The semantics are such that the assignment from one Autonomous Entity to another following the direction of the arrow.
- Data flow: This relation links a Data Prosumer to an Information Entity that is produced or consumed. This is the same relation as the Object Flow relation defined in UML and therefore the same symbol is used here.

e-bA system is complex and also spans several levels of abstraction. There are many dependencies between the levels, and these dependencies are between neighboring levels. All of the interactions feature two ways information flow. e-bA architecture must encompass autonomous, specification, roles, tasks, communications, reactive, and pro-active behavior as in figure 6. Agents also interact with their environment, through sensors and effective. The architecture must address agent mobility, translations between agent ontologies, virtual and actual migration, and centralized and decentralized collaboration.

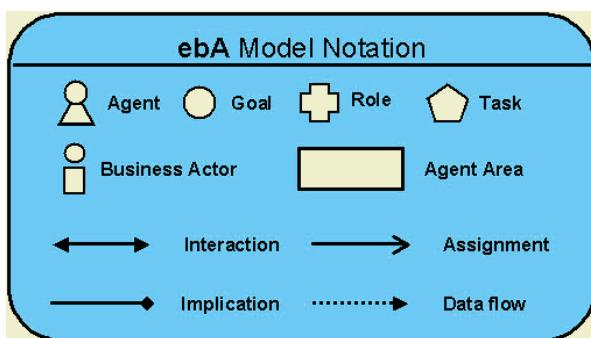


Fig. 5. ebA Model Notation.

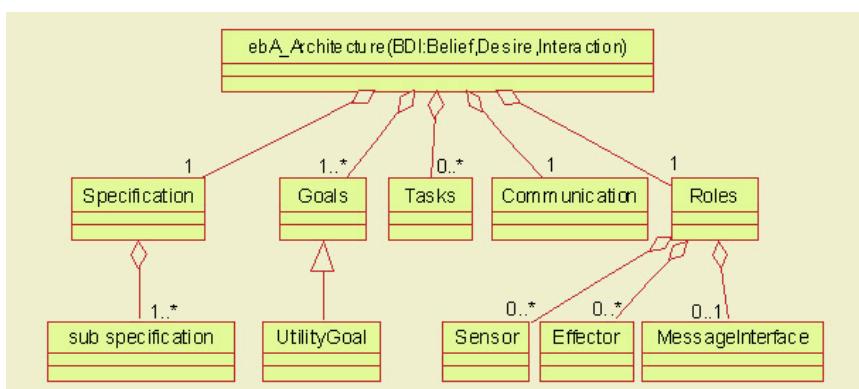


Fig. 6. ebA-Architecture.

**Table 1.** ebA-CBD reference metrics.

E-Business Agent Agent Type	System-Level Agent(00)							General Business Activity Agent(10)							Personal Agent (20)	System -Level Agent (30)	Security Agent (40)
	Interoperation Agent (01)	Business Transaction Agent (02)	Workflow Agent (03)	Marketing Agent (11)	Legal Agent (12)	Financial Agent (13)	Negotiation /Contracting Agent (14)	Information Brokering Agent (15)									
Software Agent(SWA)																	
Autonomous Agent (AJA)																	
Interactive Agent (INA)																	
Adaptive Agent (ADA)																	
Mobile Agent (MBA)																	
Coordinative Agent (COA)																	
Intelligent Agent (ITA)																	
Wrapper Agent (WRA)																	
Middle Agent (MOA)																	
Interface Agent (IFA)																	
Information Agent (IMA)																	
Smart Agent (SMA)																	
Hybrid Agent (HYA)																	
Heterogeneous Agent (HGA)																	

### 3.3.1 ebA Identification and Role Model Creation

It is mainly focuses on the individual agents and roles. It uses roles and schemes to support the diagrams for representing its characteristics such as, what goals it is responsible for, what events it needs to sense, what resources it controls, what tasks it informs how to perform, ‘behavior rules’, etc.

Figure 5 shows ebA-CBD cross reference metrics to identify the specific agent. Agent naming is referenced from use case description and role model is created using ebA model notation. ebA-CBD reference metrics also be used to define a classification code.

### 3.3.2 ebA Goal Model Creation

This model represents the Goals, Tasks, States and the dependencies among e-ba. This process is repeated for each class until no further breakdown of classes is appropriate or needed. Once all classes are defined, iterative refinement of each class is done to specify attributes, methods, and data structures. Dependencies and interactions between classes are then examined to determine what can be modeled as a component. Goals and Tasks are both associated with states so that they can be linked by logical dependencies to form graphs. It shows how the set of sub-goals after higher level goal is achieved and how Tasks can be performed to achieve the goals. A temporal dependencies can also be drawn with UML Activity Diagram notation.

### 3.3.3 ebA Communication Model Creation

Communication does not only take place between agents, but can also occur between an agent and its environment. Agents can be implemented with sensors and effectors

to allow the agent to interact with its surroundings. A less direct model of agent communication has to do with one agent effecting a change on the environment and other agents perceiving this change and acting accordingly. Instead of sending a message directly to another agent, an agent may use its effectors to change the environment in a way that can be perceived and interpreted by the sensors of other agents. This method is not as effective as message communication since it assumes that all agents will interpret this environmental change in the same manner. However, it does offer a significant advantage if message passing capability between agents is lost but coordination between agents is still required. This model highlights which, why and when agents/roles need to communicate leaving all the details about how the communication takes place to the design process. The communication model is typically refined through several iterations as long as new interactions are discovered. It can be conveniently expressed by means of a number of interaction diagrams. This model is interaction centric and shows the initiator, the responders, the motivator of an interaction plus other optional information such as the trigger condition and the information achieved and supplied by each participant.

### **3.3.4 ebA Architecture Model Creation**

A widely used agent architecture is the Belief, Desire, Intention (BDI) architecture. This architecture consists of four basic components: beliefs, desires, intentions, and plans. The exact definition of these components varies slightly from author to author, but most generally agree that all four need to be present in one form or another when using this architecture. In this architecture, the agent's *beliefs* represent information that the agent has about the world, which in many cases may be incomplete or incorrect. The content of this knowledge can be anything from knowledge about the agent's environment to general facts an agent must know in order to act rationally. This model shows agents' relationship to negotiate and coordinate in agent area. It considers the business actor and domain concept. An agent area is where software agents meet and interact in the target architecture. The agent areas can be distributed on different hosts, and facilitate means for efficient inter-agent communication.

There are two main types of agent area. as where the agents advertise their capabilities, communicate with other agents and where the user interacts with agents.

### **3.3.5 ebA Specification**

It presents ebA specification based on table 2, for the component specification, and implementation a parser to read the specification and test for the interoperation of components. The specification technique can support the identification of component interactions and serve as the basis for automatically generating test cases for the integration of components. The diagram on the bottom shows the overall structure of the tool framework. The testing framework provides both a syntactic matching scheme by looking at the signatures of the messages being sent and received, and a semantic matching scheme by generating test cases from the specification provided by each component and running them to check for interoperation. The ebA specification is based on previous models as role model, goal model, communication model and

architecture model. ebA specification is shown functional and non-functional elements as in table 2. The functional elements are described to use class diagram and sequence diagram.

**Table 2.** ebA Specification.

Item	Description
Agent Name	Identified ebA name
E-Business Type	Identified E-business Type in ebA-CBD reference architecture
General Agent Type	Identified General Agent Type in ebA-CBD reference architecture
Identification Code	Identified code in ebA-CBD metrics
Access Information	Assessed, stored and modified information
Produce Information	File/information which the agent creates while it operates
Related Agent	Agent which negotiate, coordinate or exchange message
Information Model	Represented agent's attribute and operation
Operation Model	Represented sequence of operation among agents

### 3.4 ebA-CBD Development

We have attempted summarize the process tasks into the four stages: e-business concept model, e-business static model, e-business interface identification and component spec description. The specification development takes as its input from requirements a use case model, ebA models and a ebA-spec. It also uses information about existing software assets, such as legacy systems, packages, and databases, and technical constrains, such as use of particular architectures or tools. It generates a set of component specifications and component architecture. The component specifications include the interface specifications they support or depend on, and the component architecture shows how the components interact with each other.

The identified information based on component users and performance must be provided in specification form for integration. Also, this information as in table 3 can be provided and acquired by producer, consumer and agent in interoperating system. The information of component design and development, and also functional and non-functional information must be provided by producer, and agent must provide the commercial information with this. This information is the important standard for choice and the ground for reuse to acquire the component.

### 3.5 Component Adaptation

These output are used in the provisioning phase to determine what components to build or buy, in the assembly phase as an input to test scripts.

**Table 3.** Component specification.

Item	Description
Category	Component family of Business domain
Component Diagram	Relationship between component
Component Name	Identified component name
Classification Code	Classification code of component based on ABCD Architecture
Short Description	Describe about component function, motive, constraint, etc.
Glossary	Describe concept of glossary related component specification
Component Context Diagram	Main function of Component
Component Interaction Diagram	Relationship between component
Component Sequence Diagram	Operational sequence of component
Component Diagram	Represent of required and provide interface
Component State Diagram	Represent of operation change
Interface Description	Pre/Post condition, input/output result
Usage Scenario	Scenario for component usage
Quality Attribute	Non-functional(Quality) attribute

The provisioning phase ensures that the necessary components are made available, either by building them from scratch, buying them from a third party, or reusing, integrating, mining, or otherwise modifying an existing component or other software. The provisioning phase also includes unit testing the component prior to assembly.

The assembly phase takes all the components and puts them together with existing software assets and a suitable user interface to form an application that meets the business need. The application is passed to the test phase for system and user acceptance testing.

## 4 An Example of ebA-Development on Information Search Agent Domain

We apply on ebA-CBD modeling approach to component information search agent with ebA domain requirements identification and specification development. The agent finds the information of the component to be registered newly.

### 4.1 ebA Domain Requirements for Component Information Search Agent

The component information search agent is a multi-agent, which consists of four independent agents. It is selected on ebA-CBD crossing referenced metrics as in figure 7. They collaborate for the efficient search, when a user, who is using one of the

methods, wants to get some help with another method. Component Search Agent executes the four types of search methods; keyword search, facet-based search, browsing search, and interactive search with a helper agent, as a user wants to do. In this example, we develop the keyword search agent that receives keywords from a user, and retrieves similar words from fuzzy concept relation matrix based on the relevance and relationships between concepts.

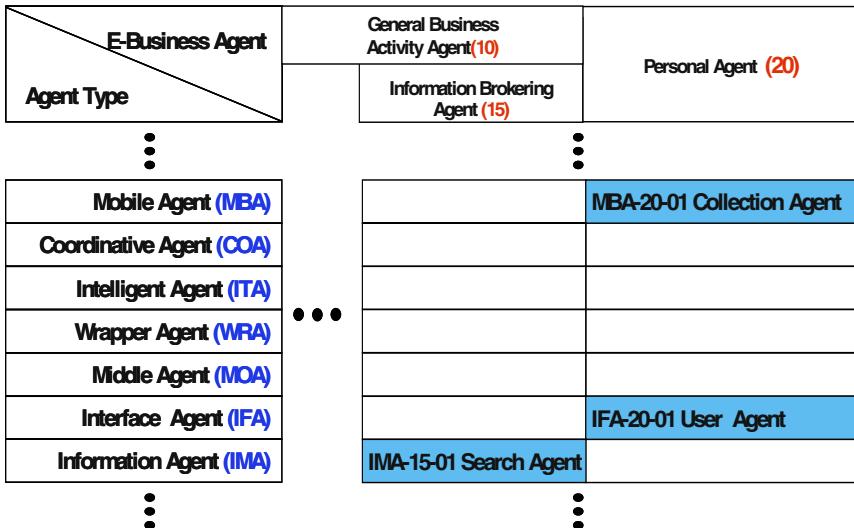


Fig. 7. ebA-CBD Metrics of Component Information Search Agent.

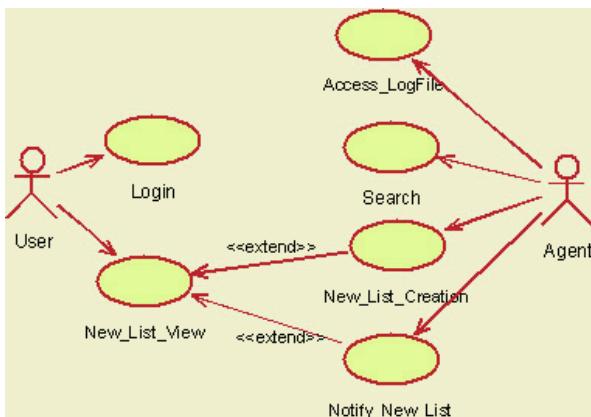
The Keyword Search Agent decides threshold based on the amount of results. The threshold is determined by the observations of the user behaviors during the search process. If there are too many search results, the Keyword Search Agent provides the user with methods to reduce the number of results. It also supports Browsing Search Agent which is conducted through the classified category tree by experts. Users retrieve the components into the category. Users can use browsing by several classifying factors, such as domain, implementation language, system type, and operating system. If the enormous results are retrieved, a user can conduct the keyword search, or the facet-based search with the results, or can request the help of the Intelligent Search Helper.

#### 4.2 e-bA Modeling of Component Information Search Agent

The e-bA modeling step is to analyze the domain and compose the requirement model into domain type model with commonality and variability. The domain modeling expresses the requirements extracted from the domain by the usecase diagram of UML.

#### 4.2.1 Use Case Diagram

The actor is extracted from the domain stakeholder and the domain external environment. The requirements of such an actor and domain operations are extracted as a domain usecase. Then a domain usecase diagram is drawn. The domain usecase is written with different levels of detail. The primitive requirements identified during the prior step should all be allocated to usecases. This provides an important link in terms of traceability between the artifacts. The domain usecase diagram should be modified to reflect the properties of domain usecase after the domain usecase generalization process. There are 5 different usecase in the ebAas in figure 8; New\_List\_View which take charge of communication with user, Search which find the information about component, and Access\_LogFile which process the access log ID and Notify\_New\_List.



**Fig. 8.** Use Case Diagram of Component Information Search Agent.

#### 4.2.2 Role and Goal Model

A domain component is extracted as an independent unit, which is the central function that provides role and goal based on usecase and actor, and satisfies mutual independence.

Figure 8 shows the overall role of component information search agent. The roles describe the external characteristics of identified agent. Figure 9 shows the main goal of the component information search agent. The Create\_New\_List goal is achieved when lower goal successfully completed. It shows how the set of sub-goals after higher level goal is achieved and how Tasks can be performed to achieve the goals.

#### 4.2.3 Communication Model

Communication model represents interactions between domain components that perform specific communication requirements. In addition, communication interface is extracted by analyzing operations between components. Communication Interaction View is presented by using an Interaction diagram, and component interface is described by using class notation as in figure 11.

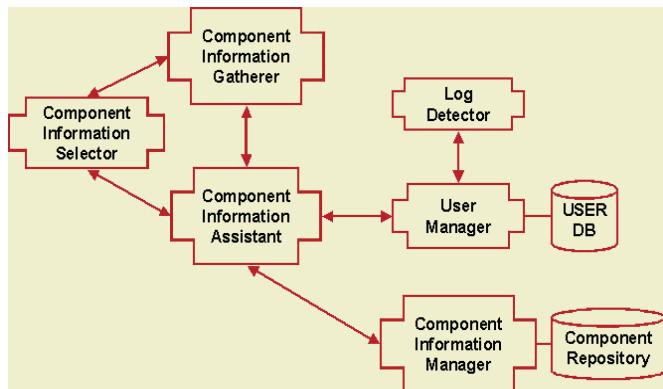


Fig. 9. Role Model.

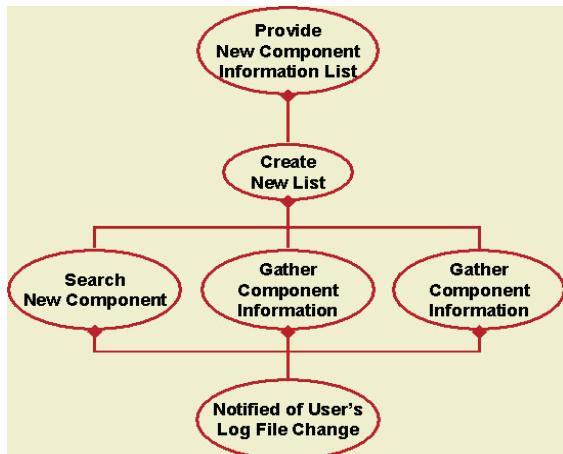


Fig. 10. Goal Model.

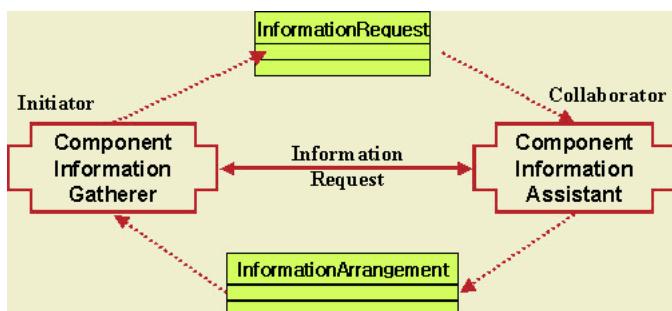
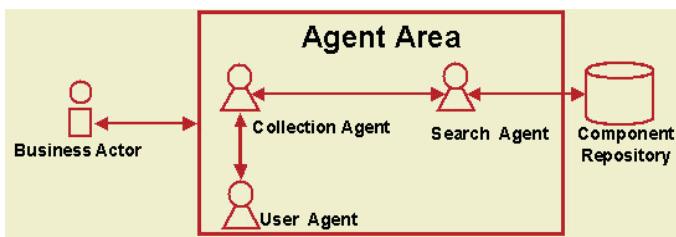


Fig. 11. Communication Model.

#### 4.2.4 Architecture Model

e-bA architecture for component information search agent presents the structure of domain components, interaction between domain components in multiple views, and specifications of the domain component. The architecture should be independent of any specific technology or set of developmental tools. The architecture should reflect properties such as commonality, variability and optional that were initialized from the requirement analysis step, refined, and maintained. Such features allow part of the architecture to be divided and replaced according to the component's property when creating the component-based software development. Figure 12 shows an architecture model. It represents overall system structure of Component Information Search Agent. Search Agent periodically update and gather to component information list and User Agent provide to alert service and manage log file.



**Fig. 12.** Architecture Model of Component Information Search Agent.

#### 4.2.5 e-bA Specification

e-bA specifications describes the purpose and interfaces of a component and furnishes information about what the component does and how it can be used. A deployable component, which is developed using a specifications, can differ in granularity according to applications. Hence, we will describe the related functions as interface and supplement required interfaces to effectively support variable granularity of the component as in figure 13. In this way when interfaces are developed independently, required interfaces can be recognized easily. The resource information is basic elements such as name, e-business type and general agent type according to ebA-CBD reference architecture and classification code. Information and operation model provide inter/external structure and interoperation of agents. These referred ebA-CBD specification development phase.

## 5 Conclusion

In this paper, we proposed ebA-CBD process and modeling method to develop e-business agent based on CBD. We also includes formal definition of proposed method as well as development of supporting tool. In defining ebA specification of whole entire agent, this method has e-business agent information more systematical and intuitive. Introducing the systematical process and ebA-CBD reference model

Item	Description
Agent Name	User Agent
E-Business Type	Personal Agent
General Agent Type	Interface Agent
Identification Code	IFA-20-01
Access Information	User Id, User Login Time, Logout Time
Produce Information	User Log File
Related Agent	Collection Agent
Information Model	<pre> classDiagram     class User {         id         passwd         name         addUser()         updateUser()         searchUser()         checkMember()         checkPasswd()         getId()     }     class Log {         LoginTime         LogoutTime         OperationInfo         getLoginTime()         getLogoutTime()         getOperationInfo()         updateLoginTime()         updateLogoutTime()         updateOperationInfo()     }     class Notify {         getNewList()         sendNewList()         checkLoginState()         checkTime()     }     User &lt; --&gt; Log     User &lt; --&gt; Notify     Log &lt; --&gt; Notify     CollectionAgent --&gt; Notify   </pre>
Operation Model	<pre> sequenceDiagram     actor User     actor Customer     actor Log     actor Notify     User-&gt;&gt;Log: Logout()     activate Log     Log-&gt;&gt;User: updateLogoutTime()     deactivate Log     User-&gt;&gt;Log: Login()     activate Log     Log-&gt;&gt;Notify: getLogoutTime()     activate Notify     Notify-&gt;&gt;User: getNewList()     Notify-&gt;&gt;User: checkLoginState()     deactivate Notify   </pre>

**Fig. 13.** ebA-Spec. of User Agent.

for e-business agent can provide the efficiency to develop it. The e-bA specification can be used as the guideline to choose desired component and be reused as based more for component creation.. This paper will reflect the features into the shape of the e-agents architecture, created a malleable architecture that can be partially separated architecture, and replaced them by the property of the components.

For the further works, the definition and detail methods should be formalized based on ebA specification. and the comparison and verification are needed though the cases study of implementation.

## References

1. Martin L. Griss, Gilda Pour, "Accelerating Development with Agent Components", IEEE Computer, pp. 37-43, May. 2001.
2. Hideki Hara, Shigeru Fujita, Kenji Sugawara, "Reusable Software Components based on an Agent Model", 7th International Conference on Parallel and Distributed Systems Workshops, 2000.
3. OMG Agent Platform SIG, "Agent Technology Green Paper," <http://www.objs.com/agent/>, 2000.
4. Q. Chen, P. Chundi, Umesh Dayal, M. Hsu, "Dynamic-Agents", International Journal on Cooperative Information Systems, 1999
5. Q. Chen, P. Chundi, U. Dayal, M. Hsu, "Dynamic-Agents for Dynamic Service Provision", Proc. of 3rd Int. Conf. On Cooperative Information Systems (CoopIS'98), 1998.
6. EURESCOMP, "MESSAGE: Methodology for Engineering Systems of Software Agents", EURESCOMP Project P907 Publication, 2000.
7. Ho-Jun Shin, Haeng-Kon Kim, "CBD Reference Architecture through E-Business Agent Classification", In Proceedings 2nd International Conference on Computer and Information Science, Aug. 2002, pp. 653-658.
8. H.K. Kim, "Component Repository and Configuration Management System", ETRI Final Research Report, 2000.
9. H.K. Kim, "Component Repository and Configuration Management System", ETRI Final Research Report, 2000.
10. H.K. Kim, E.J.Han, H.J. Shin and C.H. Kim, "Component Classification for CBD Repository Construction", In Proceeding SNPD'00, 2000, pp. 483-493.
11. Martin L. Griss, "Agent-Mediated E-Commerce Agents, Components, Services, Workflow, UML, Java, XML and Games...", In Proceedings the Technology of Object-Oriented Languages and System, Keynote Presentation, 2000.
12. Hyacinth S. Nwana, "Software Agents: An Overview", Knowledge Engineering Review, Vol. 11, No 3, pp. 1-40, 1996.
13. Nicholas R. Jennings, Michael Wooldridge, "Agent-Oriented Software Engineering", In Proceeding IEA/AIE 1999, 1999, pp. 4-10.
14. George T. Heineman, William T. Councill, Component-Based Software Engineering, Addison-Wesley, 2001.
15. Seoyoung Park, Chisu Wu, "Intelligent Search Agent for Software Components", In Proceedings Sixth Asia Pacific Software Engineering Conference, 1999, pp.154 –161.
16. Klement J. Fellner, Klaus Turowski, "Classification Framework for Business Components", In Proceeding the 33rd Annual Hawaii International Conference on System Sciences, 2000, pp. 3239-3248.
17. Nicholas R. Jennings, Michael Wooldridge, "Agent-Oriented Software Engineering", In Proceeding IEA/AIE 1999, 1999, pp. 4-10.
18. Odell, James ed., "Agent Technology", OMG, green paper produced by the OMG Agent Working Group, 2000.
19. Mike P. Papazoglou, "Agent-Oriented Technology in support of E-Business", Communications of the ACM, Vol. 44, No. 4, pp. 71-77, 2001.
20. James Odell, H. Van Dyke Parunak and Bernhard Bauer, "Extending UML for Agents", In Proceeding the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence, 2000.

21. Bernhard Bauer, Jörg P. Müller and James Odell, "Agent UML: A Formalism for Specifying Multiagent Interaction", In Proceeding 2000 Agent-Oriented Software Engineering, 2001, pp. 91-103.
22. Pearl Brereton, David Budgen, "Component-Based Systems:A Classification of Issues", IEEE Computer, Vol 33, No 11, 2000.
23. Yariv Aridor, Danny B. Lange, "Agent Design Patterns : Elements of Agent Application Design", In Proceeding Autonomous Agents 98, 1998 , pp. 108- 115.
24. Peter Herzum, Oliver Sims, "Business Component Factory", OMG press, Dec. 1999.

# Towards Efficient Management of Changes in XML-Based Software Documents

Geunduk Park<sup>1</sup>, Woochang Shin<sup>2</sup>, Kapsoo Kim<sup>3</sup>, and Chisu Wu<sup>1</sup>

<sup>1</sup> School of Computer Science and Engineering, Seoul National University  
56-1, Shillim-Dong, Kwanak-Gu, Seoul, 151-741, Korea  
[{dean,wuchisu}@selab.snu.ac.kr](mailto:{dean,wuchisu}@selab.snu.ac.kr)

<sup>2</sup> Department of Internet Information, Seokyeong University  
16-1, Jungneung-Dong, Sungbuk-Gu, Seoul, 136-704, Korea  
[wcshin@mail.skuniv.ac.kr](mailto:wcshin@mail.skuniv.ac.kr)

<sup>3</sup> Department of Computer Education, Seoul National University of Education  
161, Umyeonno, Seocho-Gu, Seoul, 137-070, Korea  
[kskim@ns.seoul-e.ac.kr](mailto:kskim@ns.seoul-e.ac.kr)

**Abstract.** Various advantages of XML such as flexibility and interoperability have given rise to a steady increase in the number of software documents using XML, and this growth has in turn necessitated new methods for systematically managing massive XML-based documents.

Unlike general documents of planar structure based on lines, documents in XML internally constitute a tree structure. Therefore, traditional version control techniques that recognize documents of planar structure are not suitable in handling hierarchically structured documents.

This paper proposes a new way of managing changes made in structured documents. While being a timestamp-based approach, the proposed method has the characteristics of maintaining version stamps for edges, rather than nodes in the tree structure, and only assigning version stamps when they are required.

## 1 Introduction

Software documents are generally produced using non-standard, proprietary editing tools, which causes heterogeneity in the format of documents, and consequently creates difficulties when sharing and referencing them [1].

In order to overcome these problems, it is essential to introduce a means that can deliver the common format to all types of software documents. XML (eXtensible Markup Language) can be an effective tool because it provides a mechanism for bridging data heterogeneity problems and has gained acceptance in the business and software development world as an open standard. When applying XML to software documentation, we can gain a number of advantages.

Firstly, the expression of software documents in a format neutral to CASE tools facilitates the sharing and exchanging of documents among development tools. Secondly, when software documents produced in each phase of a development cycle are expressed in a common format like XML, this simplifies searching and analysis of related information among the documents. Thirdly, since XML structures software documents in a

hierarchical manner, it becomes possible to utilize information on a fine-grained level with the help of technologies like XPath and XPointer. Fourthly, software documents can be treated as a knowledge base, because when necessary, it is possible to query them with query languages such as XQL, Lorel, and XQuery. Finally, as a language designed for the exchange of structured data on the internet, XML enables extensive distributed internet development.

Such advantages have given rise to a rapid increase in the number of software documents using XML, and this growth in turn made it necessary for new methods of systematically managing massive XML-based documents. Since XML documents are written, modified and shared by various developers in distributed environments, on a scale much more widespread than in general text format documents, version control techniques have become vitally important for developers to effectively grasp and manage any changes in XML documents.

The tree structure has been considered one of the most useful data structures because it is suitable for organizing data for efficient information retrieval, and representing the logical hierarchy of objects. Various techniques have been developed, attempting to manage changes made in the tree structure efficiently such as the path-copy method [2] and the fat-node method [3]. However, these methods fail to give good performances in terms of time and space required to deal with it.

This paper proposes a new version control technique, SVEM (Sparsely Version-stamped Edge Model), to manage versions of tree-structured documents. While being a timestamp-based approach, the proposed method has the characteristics of maintaining version stamps for edges, rather than nodes, and only assigning version stamps to edges that require them.

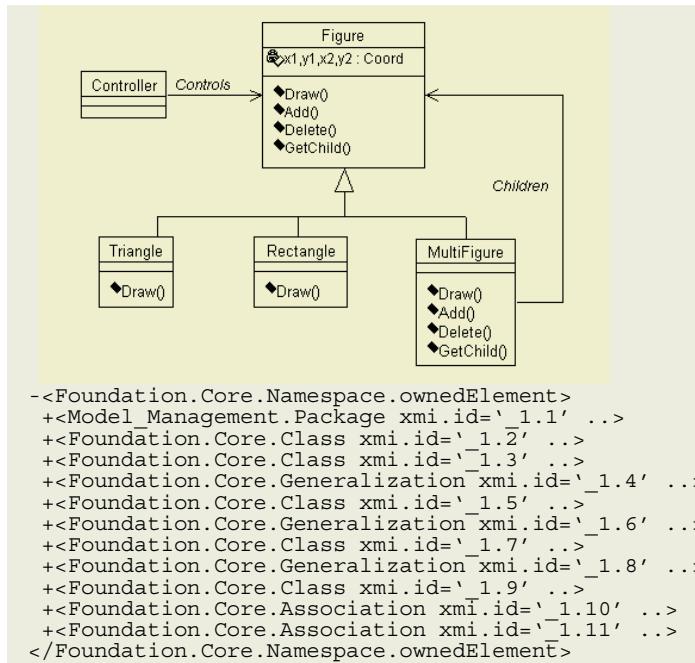
## 2 Software Documentation and XML

Several studies have been conducted and are actively underway to apply XML to software documentation. UXF (UML eXchange Format) [4] and XMI (XML Metadata Interchange) [5] are widely recognized methods of representing UML (Unified Modeling Language) modeling information in an XML format. In addition, XML is being introduced to define software for the purpose of checking consistency among software documents created in distributed environments [6] and for the purpose of simplifying software engineering analysis and achieving portable source code.

For example, figure 1 shows a UML diagram and its XMI representation. The XMI standard specifies how UML models are mapped into an XML file with a Document Definition Type (DTD). In Figure 1, the upper shows a UML class diagram representing a composite design pattern [7]. When transformed into an XMI format by an XMI tool kit [12], an XML document is generated as shown on the lower of Figure 1. XMI documents are primarily made up of two elements, `<XMI.header>` and `<XMI.content>`, and the content of a UML diagram is expressed as descendent elements of `<XMI.content>`.

The diagram in Figure 1 contains 5 classes, 3 inheritance relationships and 2 association relationships, which are expressed as children of `<Foundation.Core.Namespace.ownedElement>` in the document transformed into an XMI format.

In DOM (Document Object Model) [13] which provides an application program interface to XML data, documents are modeled as ordered-trees that are composed of



**Fig. 1.** A UML diagram and its XMI representation

various types of nodes including element nodes, text nodes, attribute nodes, entity nodes and processing instruction nodes. Accordingly, we can conceptually convert version control for XML-based software documents into version control for tree structures.

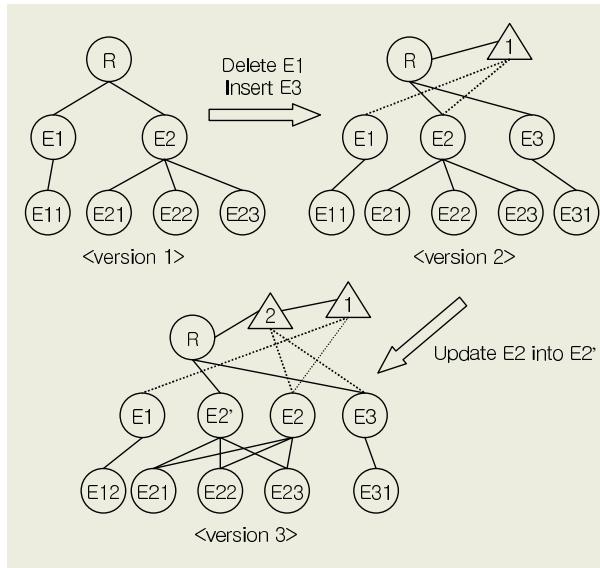
### 3 Related Work

Some version control models for tree structures, such as HiP (History in Parent) [8], RBVM (Reference-Based Version Model) [9] and SPaR (Sparse Preorder and Range) [10], have been proposed to date. HiP is a version control technique for general tree structures, while RBVM and SPaR are version control techniques for XML documents.

#### 3.1 HiP

The HiP method maintains special nodes that are called h-nodes. When a change has occurred in a certain node, an h-node is added to the parent node of the modified node, and the state immediately before the change is stored in the h-node. Figure 2 shows how versions are stored in HiP.

⟨Version 1⟩ of Figure 2 shows the initial state of the tree. When a second version is generated by deleting E1 from R, and inserting E3 into R, the h-node denoted by a triangle is added to R, a parent node of nodes where changes have occurred. Nodes E1 and E2, which were children of R before changes occurred, are stored in that h-node.



**Fig. 2.** Storing versions in HiP

And when a third version is made by updating E2 to E2', an h-node of a node R is newly added, and E2 and E3 are stored in that h-node.

This method restricts the ripple effects of the change to the parent node only. However, this is inefficient when dealing with m-ary trees that have many children like an XML tree. For example, at the worst, even when a single node out of m children has been deleted, m links to each child node have to be stored.

### 3.2 RBVM

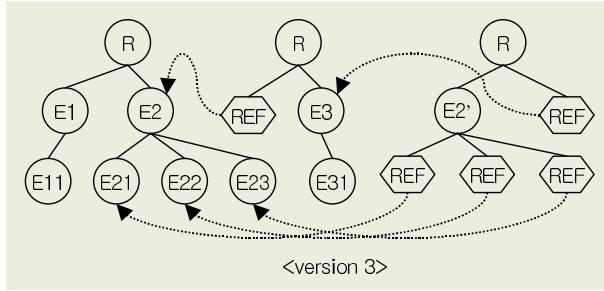
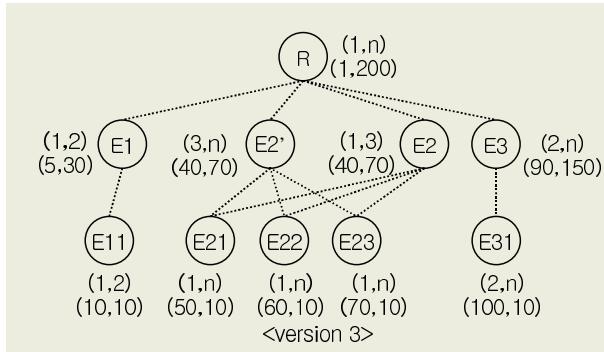
The RBVM works by pointing the reference node to the maximum sub-tree that remains unchanged from a previous version. This reference reduces redundant storage of the portion where no change has occurred.

Figure 3 shows how each version is stored in RBVM using the same example as in Figure 2. In Figure 3, nodes indicated as ‘Ref’ are reference nodes. As the maximum sub-tree is a sub-tree with E2 as a root node, a reference node only points to that sub-tree rather than storing it entirely. When a third version is made, 4 reference nodes are newly added and point to each maximum sub-tree.

While this method has an advantage of saving storage by sharing unchanged parts, it has a shortcoming in that it should maintain a lot of reference nodes in order to store previous versions. In addition, as RVBM is required to trace reference nodes backward when retrieving previous versions, it spends more time compared with HiP.

### 3.3 SPaR

SPaR maintains two pairs of numbers for all nodes. One pair of numbers consists of the creation time and the deletion time, and the other pair of numbers consists of

**Fig. 3.** Storing versions in RBVM**Fig. 4.** Storing versions in SPaR

DNN (Durable Node Number) and range. SPaR controls the ancestor-descendant relationship by utilizing DNN and range. The benefit of using DNN and range is that the ancestor-descendant relationship can be determined in constant time without physical links between them, and this mechanism also efficiently supports queries including path expressions.

In Figure 4, the upper pair of numbers represents time stamps, while the lower pair represents (DNN, range). Since E1 is inserted in version 1 and deleted in version 2, the time stamp is set as (1,2) and since E3 is inserted in version 2 and not deleted, the time stamp is set as (2, n), where n means that the node has not been deleted up to now.

Since, SPaR does not store links to children, unlike HiP and RBVM, it has an advantage in that the storage for links does not increase in proportion to the number of operations applied. However, as SPaR relies entirely on time stamps to manage versions, it should maintain time stamps for all nodes and retrieve all time stamps when retrieving specific versions.

## 4 Version Management Model

In this section, we describe our version control model, SVEM. First, we present the definition of a general tree, and then a version tree controlled by SVEM.

As a rule, a tree T consists of a set of nodes and a set of edges, where edges represent parent-child relationships between nodes. The tree with a root node r is denoted as T(r). Aside from nodes and edges, a version tree VT controlled by SVEM contains additional information. A version tree VT is defined as follows:

**[Definition 1]** A version tree VT is defined as a tuple  $\langle N, E, V, \Pi, \Omega, r \rangle$ .

N: a set of nodes

E: a set of edges

V: a set of version stamps

$\Pi$ : a set of functions from E to V

$\Omega$ : a set of operations that can be applied to VT

r: a root node

The set of edges in a version tree VT differs from the set of edges in tree T in that the former includes edges representing version relationships in addition to edges representing parent-child relationships.

All version trees are considered to have a virtual node VN, which acts as a virtual root node for the version tree VT. Accordingly, the actual root node r of a version tree becomes a child node of VN. In addition, the n-th version of a version tree is denoted as  $VT_n$  and the version tree with a root node r is denoted as  $VT(r)$ .

**[Definition 2]** A set of edges is denoted as E. E is composed of a set of parent relations ( $E_p$ ) and a set of version relations ( $E_v$ ).

**[Definition 2.1]** If a node x is a parent node of y in VT, then two nodes have a parent relation, namely,  $\langle x, y \rangle \in E_p$ . This is denoted as an edge, e(x,y).

**[Definition 2.2]** If a node i is updated to j in VT, then two nodes have a version relation, namely,  $\langle j, i \rangle \in E_v$ . This is denoted as an edge, e(j, i).

A set of children of x is denoted as  $\text{children}(x) = \{y | \langle x, y \rangle \in E_p\}$ , and a terminal node is one with no child. In other words, for a terminal node t,  $\text{children}(t) = \emptyset$  and if t is a terminal node, then the predicate,  $\text{terminal}(t)$  is true.

**[Definition 3]**  $\Pi$  is a version stamp mapping function defined as  $\Pi : E \rightarrow V$ .  $\Pi = \Pi_c, \Pi_d, \Pi_u$ , where  $\Pi_c$  maps the creation time to edges,  $\Pi_d$ , the deletion time and  $\Pi_u$ , the updating time. respectively.

A new version ( $V_{n+1}$ ) of an XML document is established by applying a number of changes to the current version ( $V_n$ ). These changes can be generated directly from the edit commands of an XML editor or can be obtained by diff packages. Numerous diff algorithms that detect deltas between versions have been developed and recent studies on diff algorithms for XML data are also in progress [11]. By utilizing such techniques, we can identify a set of operations applied to generate the next version. SVEM defines three primitive operations as follows.

**[Definition 4]**  $\Omega$  is a set of primitive operations. Three primitive operations are defined in SVEM, namely,  $\Omega = \{I, D, U\}$ , where I represents an insert operation, D, a delete operation and U, an update operation, respectively.

If we assume that each operation is performed in version v, the pre-condition and the post-condition of each operation are as follows :

**Insert.**  $I(n, T)$  inserts  $T = < N', E', r >$  into as a child of the terminal node n of  $VT = < N, E, V, \Pi, \Omega, root >$ .

(Pre-condition)  $terminal(n) = true$ .

(Post-condition)  $(N = N \cup N') \wedge (E = E \cup E') \wedge (E_p = E_p \cup \{< n, r >\}) \wedge (< e(n, r), v > \in \Pi_c)$ .

**Delete.**  $D(n, VT')$  deletes  $VT' = < N', E', V', \Pi', \Omega', r >$  from a node n of  $VT = < N, E, V, \Pi, \Omega, root >$ .

(Pre-condition)  $< n, r > \in E_p$ .

(Post-condition)  $(N = N - N') \wedge (E = E - E') \wedge (E_p = E_p - \{< n, r >\}) \wedge (< e(n, r), v > \in \Pi_d)$ .

**Update.**  $U(n, n')$  updates a node n to n' in  $VT = < N, E, V, \Pi, \Omega, root >$ .

(Pre-condition)  $< m, n > \in E_p$ .

(Post-condition)  $(N = N \cup \{n'\}) \wedge (E_p = E_p \cup \{< m, n' >\} - \{< m, n >\}) \wedge (E_p = E_p \cup \{< n', k > | k \in children(n)\}) \wedge (E_p = E_p - \{< n, k > | k \in children(n)\}) \wedge (E_v = E_v \cup \{< n, n' >\}) \wedge (< e(n', n), v > \in \Pi_u)$ .

**[Definition 5]** The stack for storing version stamps is called the stack for version stamps, VStack, and each record stored in the VStack is defined as a tuple (v, e, t).

v : the version stamp

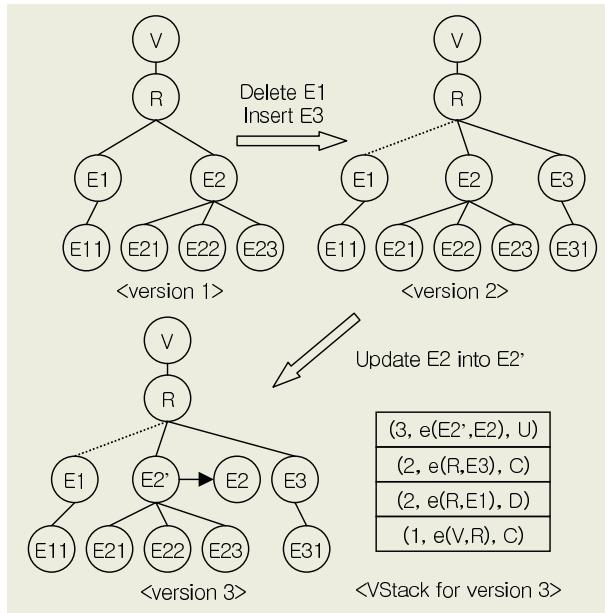
e : the edge that a version stamp is assigned to

t : the type of the version stamp

When a version stamp is assigned by  $\Pi_c$ , t has a value of C, when assigned by  $\Pi_d$ , D, and when assigned by  $\Pi_u$ , U, respectively. Since version stamps are not assigned to all edges in SVEM, they are stored in a stack structure, rather than in edges themselves, so as to efficiently access to edges to which version stamps are assigned. For example, when the edge  $e(n, r)$  is deleted at version v,  $\Pi_d(e(n, r)) = v$ , and this version stamp is stored in the VStack in the form of (v, e(n, r), D).

As XML documents are ordered trees, it is necessary to preserve the order of a child node. We can preserve the order by extending the format of records stored in the VStack to (v, e(n, r), t, k) where k means a node r is the k-th child of a node n. However, for simplicity, we will omit this without loss of generality.

Figure 5 shows how SVEM stores versions with regard to the same example as in Figure 2. When the first version is made, a version stamp is given to the edge  $e(V, R)$ , namely,  $\langle e(V, R), 1 \rangle \in \Pi_c$ . In the second version, the deletion time is given to the edge



**Fig. 5.** Storing versions in SVEM

$e(R, E1)$  by deleting the sub-tree  $VT(E1)$  and the creation time is given to the edge  $e(R, E3)$  by inserting the sub-tree  $T(E3)$ . In the third version, since  $E2$  is updated to  $E2'$ ,  $E2'$  is inserted into the place of  $E2$  and the edge  $e(E2', E2)$  is created, namely,  $\langle e(E2', E2), 3 \rangle \in \Pi_u$ .

## 5 Version Retrieval

In SVEM, retrieval of versions can be made by sequentially reading records stored in the VStack, and by performing the rollback operation  $R$  that adjusts invalid edges. Each record in the VStack consists of a 3-tuple  $(v, e, t)$  and conditions where an edge should be adjusted in  $VT_x$ , version  $x$  of the version tree  $VT$ , are as follows:

- $(t = C) \wedge (x < v)$

Since a record in the VStack,  $(v, e(n, r), C)$  means the creation time of  $e(n, r)$  is after time  $x$ ,  $e(n, r)$  is invalid in  $VT_x$ . Therefore, the  $R$  operation which deletes  $e(n, r)$  is performed and as a result of this operations, we can consider the sub-tree  $VT(r)$  to be deleted because paths to all descendant nodes of  $r$  are deleted.

- $(t = D) \wedge (x < v)$

Since a record in the VStack,  $(v, e(n, r), D)$  means the deletion time of  $e(n, r)$  is prior to time  $x$ ,  $e(n, r)$  has to be restored to its original position in  $VT_x$ . Therefore, the  $R$  operation which restores  $e(n, r)$  is performed, which results in the effect of the sub-tree  $VT(r)$  being inserted.

- $(t = U) \wedge (x < v)$

Since a record in the VStack,  $(v, e(n'), n, U)$  means the creation time of  $e(n', n)$  is after time  $x$ ,  $e(n', n)$  is invalid in  $VT_x$ . Therefore, the R operation which restores  $n'$  to its previous state, a node  $n$ , is performed, which causes  $n$  to be moved to the position of  $n'$  and all parent relations of  $n'$  to be transformed into parent relations of  $n$ .

The algorithm Retrieve() retrieves an arbitrary version,  $VT_x$ , from the current version  $VT_n$ . The algorithm consists of two phases, the first being the retrieval phase that constructs a target version, the second being the restoration phase that returns it to the current version. In the retrieval phase, rollback operations are applied by reading records that satisfy  $v > x$  from the top element in the VStack. When such rollbacks are completed, the tree is copied along with parent relations, disregarding version relations. The resulting tree is the version tree to be retrieved.

Since the version tree  $VT_n$  has been changed in the process of retrieving a version, operations that return it to the current version are carried out in the restoration phase. In this phase, records in the VStack is read from  $v = x + 1$  in the direction opposite to that of the retrieval phase and an reverse operations of R are applied depending upon the type of version stamps.

```

Tree Retrieve(VT T, int x)
    // T : the version tree controlled by SVEM.
    // x : the version number to retrieve.
Tree Tx;
element = T.stack.getFirstElement();
while(element.ver()>x) {
    edge = element.edge();
    switch (element.type()) {
        case C : disconnect(edge);
        case D : connect(edge);
        case U : shiftLeft(edge);
    }
    element = T.stack.getDownward();
}
Tx = T.copyTreeFollowingParentRelation();
while (element!=NIL){
    edge = element.edge();
    switch (element.type()) {
        case C : connect(edge);
        case D : disconnect(edge);
        case U : shiftRight(edge);
    }
    element = T.stack.getUpward();
}
return(Tx);

```

## 6 Performance Analysis

In this section, we present the performance analysis of HiP, the fully time-stamped approach and SVEM in dealing with m-ary tree. We define a transaction as a set of operations to be performed to generate the next version from a current version and assume the average number of operation in one transaction is  $t$ . A comparison of performance of each method is given below:

### 6.1 Time and Storage Required for Each Transaction

When a change occurs in a child node, HiP should store all previous children in the h-node of its parent in order to maintain the previous state. So if a node has  $m$  children on the average, the time and space needed to process for an operation are proportional to an average number of children,  $m$ . Consequently, the processing time and the storage required for each transaction is  $O(tm)$ .

In the fully time-stamped approach, because version stamps should be maintained for all nodes, the time and space needed to process for an operation is  $O(tn)$  where  $n$  is an average number of nodes in each operation. Generally  $n$  is greater than  $m$  because I and D operations are carried out with a sub-tree as a unit.

In contrast, SVEM only requires version stamps for changed nodes to be stored in the VStack, so an operation can be performed in constant time and only one record is added in the VStack. Consequently, the processing time and the storage required for each transaction is  $O(t)$ .

### 6.2 Retrieval Time for Old Versions

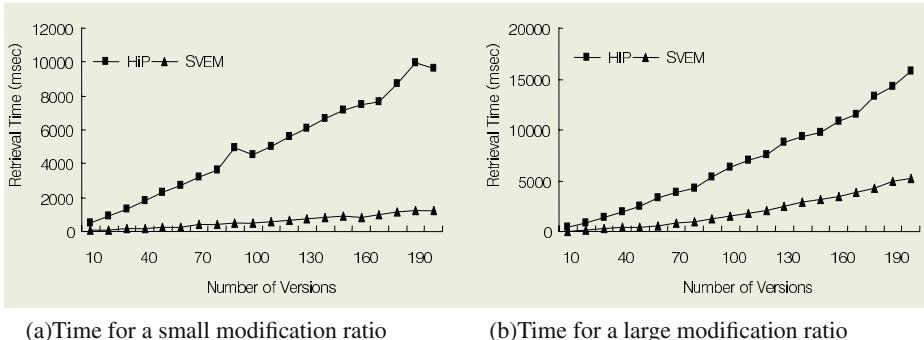
In HiP, the total number of h-nodes in the version tree is  $vt/c$  where  $v$  is the number of versions, and  $c$  is the average number of changed children of the same parent within one transaction. By definition, the range of  $c$  is from 1 to  $m$  in normal cases. In addition, because, for each child node of an h-node, we should check whether it has its own h-node,  $vtm/c$  nodes should be checked in the worst case to retrieve a particular version. In other words, the retrieval time is  $O(vtm/c)$ .

In the fully time-stamped approach, if  $n$  is the average number of nodes in each operation, the total number of nodes that belong to the current tree is  $vtn$ . As all the nodes in the current tree should be checked for reconstructing a certain version in the worst case, the retrieval time is  $O(vtn)$ .

In SVEM, if  $t$  operations occur in each transaction, there are  $vt$  records in the VStack. Therefore,  $vt$  records should be examined in each phase of SVEM, and the retrieval time is  $O(vt)$ .

## 7 Experimental Results

To evaluate the performance, we measured retrieval time of HiP and SVEM and results are shown in Figure 6. This experiment has been conducted on a dual-Pentium III 500 MHz using a version generator. The version generator generates I, D, and U operations



**Fig. 6.** The Comparison of the Retrieval Time between HiP and SVEM

using a random function and can be controlled by some parameters such as the number of versions, the number of operations in one transaction.

The parameters used in the simulation are as follows: The average number of nodes in the initial tree is 10 000, the average number of children is 10, and the number of versions generated is 200. Figure 6 (a) gives the retrieval time of each method for documents with a small modification ratio, namely with 20 operations in one transaction on the average and Figure 6 (b) gives the retrieval time for documents with a large modification ratio, namely with 200 operations.

SVEM retrieves versions 8~10 times as fast as HiP for a small modification ratio and 3~8 times for a large modification ratio. When a small part of documents is modified, the possibility is very high that one h-node is added for every single operation in HiP.

In contrast, when documents are modified largely, as the possibility gets higher that multiple children of the same parent are changed in one transaction, it is sufficient to add only one h-node for several operations. That is the reason why the ratio of the retrieval time grows smaller for a large modification ratio and implies there is room to enhance SVEM by using optimizing techniques and heuristics that can reduce the amount of information on version stamps.

## 8 Conclusion

Software documents are increasingly being expressed using XML, as the vast benefits of this markup language become apparent. Unlike software documents of planar structure, software documents in XML can be modeled as m-ary trees. This paper proposed an SVEM method that has advantages in terms of time and space in dealing with m-ary trees compared with existing version control techniques for the tree structure.

However, there is room to improve SVEM. We are conducting further research on optimizing techniques and heuristics that can compress records and consequently reduce the number of records in the VStack.

## References

1. David Mundie, Using XML for Software Process Documents, In Proc. of the Workshop on XML Technologies and Software Engineering (XSE2001), 2001.

2. C. W. Fraser and E. W. Myers, An Editor for Revision Control, ACM Transactions on Programming Languages and Systems, 9(2), 277-295, April 1987.
3. J. R. Driscoll, N. Sarnak, D. D. Sleator and R. E. Tarjan, Making Data Structures Persistent, Journal of Computer and System Sciences, 38, 86-124, 1989.
4. Junichi Suzuki and Yoshikazu Yamamoto, Making UML Models Exchangeable over the Internet with XML: UXF approach, In Proc. of the First International Conference on the Unified Modeling Language (UML '98), pp. 65 - 74, Mulhouse, France, June 1998.
5. Object Management Group, OMG-XML Metadata Interchange (XMI) Specification, v1.2, January 2002.
6. Nentwich, C., Emmerich, W. and Finkelstein, A. Static Consistency Checking for Distributed Specifications, In Proc. of Automated Software Engineering 2001, San Diego, 2001.
7. Gamma E.,et al. DesignPattern : Elements of Reusable Object-Oriented Software, Addison Wesley, 1995.
8. E. J. Choi, Y. Kwon, An Efficient Method for Version Control of a Tree Data Structure, Software : Practice and Experience, vol.27, no 7, pp.797-811, July 1997.
9. S.-Y. Chien, V.J. Tsotras, and C.Zaniolo, Efficient Management of Multiversion Documents by Object Referencing, In Proc. of the 27th VLDB, Rome, Italy, 2001.
10. S.-Y. Chien, V.J. Tsotras, C.Zaniolo, D. Zhang, Storing and Querying Multiversion XML Documents using Durable Node Numbers, In Proc. of the 2nd International Conference on Web Information Systems Engineering, 2001.
11. G. Cobena, S. Abiteboul, and A. Marian. Detecting Changes in XML Documents, In Proc. of the 18th International Conference on Data Engineering, 2002.
12. IBM alphaWorks, See <http://www.alphaworks.ibm.com/tech/xmitoolkit>.
13. Document Object Model (DOM) Level 1 Specification Version 1.0. See <http://www.w3.org/TR/REC-DOM-Level-1>, 1998.

# Model-Based Project Process Analysis Using Project Tracking Data

Kyung-A Yoon, Sang-Yoon Min, and Doo-Hwan Bae

Software Process Improvement Center, Department of Computer Science  
Korea Advanced Institute of Science and Technology  
373-1, Kusong-dong, Yosong-gu, Taejon 305-701, South Korea  
<http://se.kaist.ac.kr>

**Abstract.** In software process improvement, accumulating and analyzing the historical data from past projects are essential work. However, setting up the systematic and logical measurement and analysis program is very difficult. Many mature organizations have their own measurement program for the process improvement. However, most of them are based on the statistical metrics-driven approach that consequently limits logical reasoning on the detailed analysis on the process. In this paper, we propose a process analysis approach called MPAF(Model-based Process Analysis Framework), based on formal process modeling. In MPAF, the corresponding formal process instance model is recovered through data gathering from a project execution. Various formal analysis can be performed on the recovered and reconstructed process instance model for diagnosing the vitality of the project. We also performed experimental case study by applying MPAF to real world industry projects.

## 1 Introduction

To reduce the development cost and time, software organizations want to exactly predict and to steadily improve the quality of software and its development productivity. Because software process is only one of a number of “collectable factors in improving software quality and organizational performance”[1], the quality and productivity are influenced by the software processes used for executing a software project. In the industrial fields, there already has been great effort in improving software process such as the assessment model forms, CMM and SPICE. Many software organizations try to improve software processes by making the gained knowledge from past projects persistent and thus reusable for future projects.

For the purpose of obtaining and manipulating the knowledge from projects, product-oriented metrics such as LOC, function point, number of errors are generally used in the project and process analysis. However, the methods using such product-oriented attributes do not completely support the analysis of process because they lack in viewpoints of process elements such as analysis based on activity, artifact, and agent. To make up for these weak points in the product-oriented metrics, we consider the analysis methods based on the process model which visualizes the unified infor-

mation and provides the viewpoints of process components. Since the process model can reduce ambiguity, it can improve the understandability of process, reduce the data complexity for analyzing, and support automation of analysis tasks through PSEE(Process-centered Software Engineering Environment).

In this paper, we develop a Model-based Process Analysis Framework(MPAF) in order to analyze, assess, and improve the software processes, and conduct an empirical analysis with 10 projects. MPAF provides the analysis elements and the procedure, both of which are used to construct the process model instance using the extracted information from executed project data and analyze it with the defined analysis factors. Through MPAF, we can analyze the complex process with the high level of understanding, support to easily find the points needed to improve at the level of process elements, and automate the analysis procedure in PSEE. As the target process environment and process modeling language, we have adopted SoftPM(Software Process Management system)[2], CogNet[2], and AG(Artifact Graph)[3].

The remainder of this paper is organized as follows. The next section gives a brief description of SoftPM, CogNet, and AG. In Section 3, we discuss the overview, key elements and procedures of MPAF that we propose in this paper. In Section 4, the definitions and usages of some measures for the more detailed analysis are represented according to the analysis factor. The empirical study that is carried out with industrial data is described in Section 5. Section 6 presents the related work to our approach. Finally, Section 7 draws the conclusion with some future work.

## 2 A Brief Introduction to SoftPM, CogNet, and AG

### 2.1 SoftPM

SoftPM is a software process design and management system, which supports to model, enact, monitor, and actually execute a process with a high level Petri-net formalism[2]. It also offers a variety of managerial analysis techniques in addition to the conventional Petri-net analysis. To take advantage of the complex modeling formalism and reduce the difficulties of understanding it, SoftPM provides the multi-level modeling capability which supports three different levels for process representation. They are the cognitive, MAM-net and Pr/T-net levels. Each level has different roles in SoftPM: the cognitive level acts as the primary modeling layer, the MAM-net level as the core process representation formalism and the Pr/T-net level as the low-level base formalism. To interact with a process designer, the cognitive level provides CogNet modeling language that is discussed in the next subsection.

### 2.2 CogNet

In general, a process model is constructed to increase the understandability of a process, to reduce the difficulty of validating a process, and to automate on PSEE. CogNet is a process modeling language which is used in the cognitive level of SoftPM with

activity, artifact, and agent aspects[2]. It consists of two constructs, of which one is a process activity and the other is an artifact flow. A process activity is denoted by a rectangle and an artifact flow is denoted by an arc. CogNet is defined as follows:

**definition (CogNet)** *CogNet* is the finite set of *AN* and *AF*:

- *AN* : the finite set of nodes representing the process activities in the cognitive model,
- *AF* : the finite set of direct arcs representing the artifact flow in the cognitive model such that  $((AF \subset (AN \times AN)) \wedge (AF \text{ is not reflexive}) \wedge (AF \text{ is antisymmetric}))$ .

The following attributes are mainly used to the analysis tasks among the properties of the process activity:

- *processName* : unique name for each process activity
- *preRelation* and *postRelation* : the relationships with other activities in terms of the execution order and the directions of the artifact transfer
- *expectedDuration* and *actualDuration* : expected and actual duration time
- *startDate* and *endDate* : actual start and end date
- *expectedStartDate* and *expectedEndDate* : expected start and end date
- *assignedAgents* : the list of the human agents who are assigned to the activity

### 2.3 AG(Artifact Graph)

SoftPM provides an artifact graph(AG), which describes the artifacts and their relations[3]. Since these relations present the order of producing artifacts and the relation of referencing among artifacts, it can show artifacts that give effects to change or usages to produce other artifacts. In this paper, we use it for evaluating the artifact-oriented compliance of an actual process according to a project plan. An artifact is denoted by a rectangle and a reference relation is denoted by an arc. AG is defined as follows:

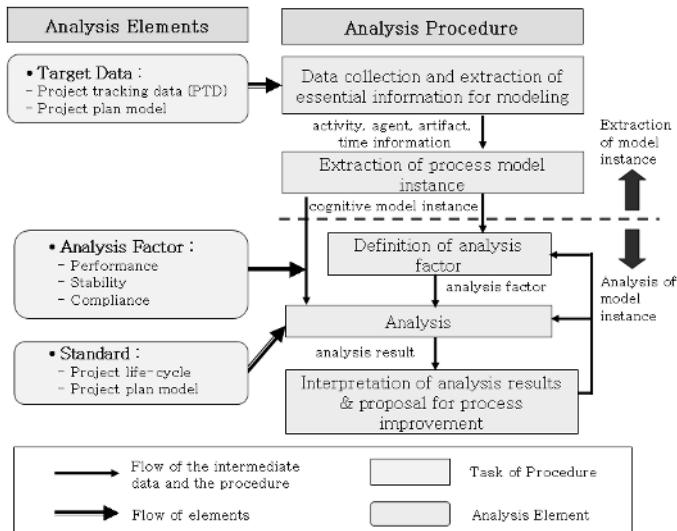
**definition (Artifact Graph)** *Artifact Graph(AG)* is the finite set of *R* and *F*:

- *R* : the finite set of artifact,
- *F* : the finite set of arcs representing the reference relation.

## 3 MPAF(Model-Based Process Analysis Framework) Using Project Tracking Data

In this section, we give an explanation on Model-based Process Analysis Framework(MPAF). As depicted in Fig.1, MPAF consists of the analysis elements and the

analysis procedure. The analysis elements are the sets of information which are used in analysis and diagnoses. The analysis procedure is composed of the tasks of extracting the process model instance and analyzing it. The following subsections provide the detail description on the analysis elements and the analysis procedure.



**Fig. 1.** The overview of MPAF.

### 3.1 Analysis Elements

According to its usage, the analysis elements are classified as the target data for a process model extraction, the analysis factor for a process analysis, and the standard for a process validation.

**Target Data.** In order to analyze the process through MPAF, we need the process model instance of the target project at first. For extracting the process model instance, MPAF uses the project tracking data as the execution data of the target project.

Project tracking is an integral part of project management that is required at CMM level 2[4]. It needs to capture the current state of the project and provide the proper control to the project. Therefore, Project Tracking Data(PTD) is the information on the actual state of the project including the basic data for software engineering that consists of effort, schedule, defects and size[5]. Once a detailed project plan is made, in which all schedulable tasks are listed along with their start and end dates and the assigned person, at the very least the execution of the tasks must be tracked.

**Analysis Factor.** Analysis factor is defined from the viewpoint of process analysis to be used to show the characteristics of process. There are four main issues on the road to process improvement: performance, stability, compliance, and capability of process[6]. Among these issues, the analysis factors of MPAF include performance, sta-

bility, and compliance except capability that is difficult to measure using only the process model. Each analysis factor has the measures of its own.

- Performance

Performance is the actual result that can be acquired when software organization observes the software process. Particularly, it is the actual ability of a process within the development cost and schedule required by the customer. In this paper, we only consider the time-related aspects of performance and classify this with two categories, which are the activity-oriented performance and agent-oriented performance. The supported measures of activity-oriented performance are the delayed days of activity, idle days of activity, and concurrency degree of activity. The measures of agent-oriented performance are the average number of the agent conflict activity and average time of the agent conflict.

- Stability

To control the process, the stability about the variance of process is needed. MPAF provides the concept of the process pattern to evaluate the stability of a process. Contrary to Ambler's definition that is the collection of techniques and activities for developing object-oriented software[7], the process pattern in our approach is defined as follows:

**definition (Process pattern)** *Process pattern* is the set of repeatable and predictable tendencies of the development process. It is categorized into the structural pattern and behavioral pattern.

- Structural pattern

It is defined as the set of general activities(artifacts) and their relations according to development process(es) executed in the common conditions such as in the same software organization or by the same development team. The activity-oriented pattern uses CogNet and the artifact-oriented pattern uses AG.

- Behavioral pattern

It is defined as the behavioral tendency of activities of the process(es) and analyzed with the managerial aspects. The examples of behavioral pattern are the tendency of activities' concurrency, the tendency of idle-time interval and duration, and so on.

- Compliance

The process compliance depends on the reliable process behavior. Therefore the degree of compliance means that processes are faithfully executed in the standard way. In this paper, we mainly consider the executed order of activities and the produced point in time of artifacts against the project plan as compliance. The provided measures in MPAF are the activity-oriented and artifact-oriented compliances.

**Standard.** A standard, such as international standard, basic process of common methodology, project plan, plays an important role for validating and analyzing a target project process. MPAF uses the project plan and the software life-cycle as the standard for process validation. To easily evaluate the degree of compliance with a project plan, it is needed that the standard is modeled using CogNet and the abstraction levels of the standard model and the actual process model instance are adjusted to the same level.

### 3.2 Analysis Procedure

The analysis procedure is divided broadly into two subprocedures: extracting procedure and analyzing procedure. The former consists of the three tasks which are the data collection, the extraction of essential information for constructing a CogNet model from PTD, and the recovery of the process model instance. The latter consists of another four tasks which are the decision of the analysis factor, the analysis of the process model instance with analysis factor, the interpretation of analysis results, and the proposal for process improvement.

**Extraction of the Process Model Instance.** If the process is modeled before its execution, the extraction of the process model instance is a kind of reverse engineering in the process research area. System reverse engineering candidates usually exhibit a variety of undesirable characteristics, such as poorly structured source code, missing or incomplete design specifications, or modules incorporating extreme complexity[8]. Through the recovery of the process model instance, we can easily understand the actual complex process and find the problems in it. To recover the model instance, we extract the activity, agent, artifact, and schedule information as shown in Table 1 from PTD as the essential information.

**Table 1.** The essential information from PTD for model construction.

Classification	Essential information	Remark
Schedule tracking (activity-related)	Activity (name of activity) Actual start/end date	
Agent-tracking (agent-related)	Agent name Activity (name of activity)	If the “type” field has “actual” value carried by agent
Artifact (document)	Title Actual date for measuring the document size	Artifact name

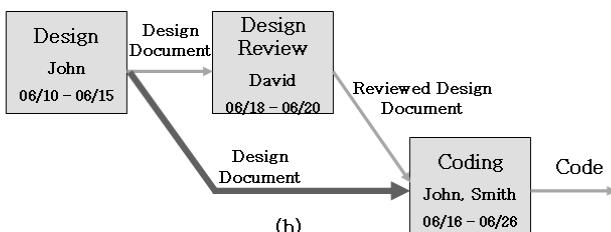
The process model recovery and the system design recovery have similar problems related to the use of incomplete or ambiguous source information, specially PTD in our approach. The types of problems that can be happened during model recovery are summarized as follows, including PTD-related problems.

- Cases related to PTD
  - Provision of insufficient information :  
The dependency relationship between activities doesn't exist, and so on.
  - Provision of incomplete information :  
The artifact information exists but the information of activity that produces this artifact dose not exist, and so on.
- Case related to the process modeling language  
The problems which are caused by the expression power of CogNet

From these essential information in Table 1, we construct the process model instance with CogNet. The following procedure is the steps of the model recovery. We present it very simply in this paper. However this procedure can be described in detail by pseudo code based on the graph formalism, and automated in PSEE. To help the understanding of this procedure, Fig. 2 shows the example of extracting the process model instance.

	Essential information for modeling		Example		
	name	Design	Design Review	Coding	
Activity	actual start date ~ actual end date	06/10 ~ 06/15	06/18 ~ 06/20	06/16 ~ 06/26	
Agent	name	John	David	Smith	
	activity name	Design, Coding	Design Review	Coding	
Artifact	title	Design Doc.	Reviewed DD	Code	
	actual date of measuring the # of pages	06/15	06/20	06/26	

(a)



(b)

**Fig. 2.** The example of extracting the process model instance : (a) the extracted essential information from PTD, (b) the extracted process model instance.

**Step 1.** Create the activity node, and then insert the activity, artifact, agent and schedule information of the essential information of PTD. In the example of Fig.2, the three activities, which are design, design review, and coding activity, are created with the related artifact and agent information of the essential information(a) that is extracted from PTD.

**Step 2.** Create the arc(artifact flow) between activities according to the project plan model, and then insert the arc information containing the artifact name. If the artifact information exists but the activity information which produces this artifact does not, create the virtual activity that is named as artifact name added after “create\\_”. In the example of Fig.2, the artifact flows which are the direct arcs from design to design review activity and from design review to coding activity are created according to the project plan.

**Step 3.** Sort the activities by the actual start day and actual end day order. During this work, the new relations between activities may be created. According to this new relation, create properly the new arc. In the example of Fig.2, the new relation is created(represented by the bold direct arc in Fig.2). After sorting the three activities by the actual start and end dates, we can easily find that the actual start date of the coding activity was earlier than that of the design review activity. This case implies that the coding activity began as soon as finishing the design activity and took the reviewed design document from design review activity during execution. Therefore the new artifact flow from design to code is created.

**Analysis of the Process Model Instance.** We describe the analysis of the process model instance with the analysis factors and their measures as shown in Table 2. The analysis scale is divided into two categories: single project, and multiple projects. The process analysis of “single project” has the purpose of evaluating the process of a specific project. The process analysis of “multiple projects” leads to the success of future projects by predicting the problems since it can provide the structural and behavioral tendencies of the process. In the section 4, we present the definition of the measures related to each analysis factor.

**Table 2.** The analysis factor of MPAF.

Scale	Analysis factor		Measure
Single project	Performance	Activity-oriented	- Delayed days of activity - Idle days of activity - Concurrency degree of activity
		Artifact-oriented	- Average frequency of the agent conflict - Average time of the agent conflicts
		Compliance	- $Compliance_{ACP}$ Deviation $_{ACT}$ - $Compliance_{ART}$ Deviation $_{ART}$
	Stability	Activity-oriented	- Dependency between activities and artifacts
		Behavioral pattern	- Concurrency tendency of activity - Tendency of idle-time interval and duration - Tendency of frequently inserted and deleted activity and artifact

## 4 Detailed Analysis of the Process Model Instance Using MPAF

In this section, we discuss the definitions and usages of some measures for the more detailed analysis according to the three analysis factors, which are the performance, compliance, and stability.

### 4.1 Performance

In our approach, we consider the performance of the process as the efficiency of the activity and agent with the time aspect of the performance.

**Activity-Oriented Performance.** The efficiency of the activity means whether each activity adheres to its' planned schedule or not. It has the following measures:

- Delayed dates

Whenever the duration of delayed dates is longer, the efficiency of the activity is lower. The duration of the delayed dates per activity ( $Activity_{Delay}$ ) is defined as follows( $AN$  is an activity node in the CogNet process model instance):

$$Activity_{Delay} = AN的实际持续时间 - AN.预期持续时间 \quad (1)$$

- Idle and overlapped dates per activity

In the actual process, the case exists in which the next activity begins in spite of the fact that the previous activity is not completed, and vice versa. The latter is the idle dates and the former is the overlapped dates defined as in the following equations( $AN_{To}$  represents the following activity and  $AN_{From}$  represents the previous activity in the same CogNet process model instance):

$$Date_{overlapped} = AN_{To}.startDate - AN_{From}.endDate \quad (2)$$

- $Date_{overlapped} < 0$  : whenever overlapped date is longer, the total project execution date is not always shorter.
  - $Date_{overlapped} == 0$  : ideal case is that the following activity begins as soon as the previous activity is finished.
  - $Date_{overlapped} > 0$  : idle dates
- Concurrency of the activity

The concurrency of the activity means the number of activities having the partially overlapped or the same execution day of this activity, and is defined as follows ( $Act$  and  $Activity_i$  are activity nodes in the same CogNet process model instance):

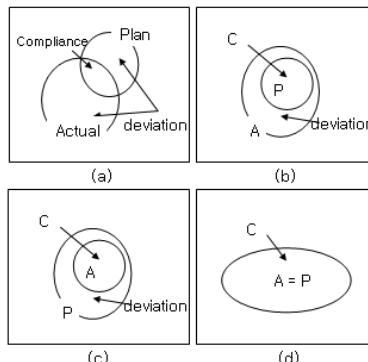
$$\begin{aligned}
 \text{Concurrency}(Act) = & \\
 \#\{\text{Activity}_i \mid (\text{Act.startDate} < \text{Activity}_i.\text{endDate} \leq \text{Act.endDate}) \\
 & \vee (\text{Act.startDate} \geq \text{Activity}_i.\text{startDate}) \\
 & \wedge (\text{Act.endDate} \leq \text{Activity}_i.\text{endDate})) \\
 & \vee (\text{Act.startDate} \leq \text{Activity}_i.\text{startDate} < \text{Act.endDate}))\}
 \end{aligned} \tag{3}$$

**Agent-Oriented Performance.** In MPAF, the agent-oriented aspects of the performance is considered with the two measures related to the agent conflict. Many actual processes have the agent conflict problems. It means that the same agent is assigned to the multiple activities which share the overlapped dates. While the agent conflict isn't expected in the project plan, generally, most delayed projects which have the resource constraint show such tendency to overcome the time delay with the job concurrency. Because the agent conflict gives an effect to the process performance and the quality of the work result, it should be monitored and controlled. From Eqs.(4) to (5), *Agent* is defined as the set of agents in the specific process and  $AG_i \in Agent$ . Thus, operator # means "number of" and the type of variable i is integer.

- Average frequency of the agent conflicts

The average frequency of the agent conflicts means how many the agent conflicts are occurred to the one agent on the average. Therefore, it is defined as follows:

$$AC_{ACTFrq} = \frac{\sum_{i=0}^{\#Agent} \# AG_i.equConflictActivity}{\#Agent} \tag{4}$$



**Fig. 3.** The process compliance: (a)The actual process is partially executed along the project plan with the extra process, (b)The actual process is completely executed along the project plan but is executed with the extra process, (c)The actual process is partially executed along the project plan, (d)The complete compliance between the actual process and project plan.

$AG_i.equConflictActivity$  means the set of equivalence activity sets which have the activities of the shared overlapped dates in the activities that are executed by  $AG_i$ .

- Average time of the agent conflicts

The average time of the agent conflicts means the average time which the one agent may work in the conflict state, in other words, he may work at least two or more jobs concurrently. Therefore, it is defined as follows:

$$AC_{Time} = \frac{\sum_{i=0}^{\#Agent} AG_i.totalCFTime}{\#Agent} \quad (5)$$

$AG_i.totalCFTime$  means the total work time of  $AG_i$  in the conflict state. Its value can be acquired during constructing  $AG_i.equConflictActivity$  set.

## 4.2 Compliance

The compliance with the project plan is classified with the aspects of the activity and artifact. The activity-oriented compliance means the consistency of activities and their relations of the artifact flows between the actual process and project plan, in other words, the extracted actual process model and project plan model. Similarly, the artifact-oriented compliance means the consistency of artifacts and their reference relations between the actual process and project plan. In addition, MPAF considers the measure of the deviation that is an opposite concept of the compliance. It is classified with the aspects of the activity and artifact, too.

**Activity-Oriented Compliance and Deviation.** Fig.3 shows the four cases of the process compliance. According to the definition of CogNet, we can define the compliance and deviation using the activity nodes and their artifact flows. At first, the degree of the activity-oriented compliance( $Compliance_{ACT}$ ) is defined as how many activity nodes and artifact flows of the actual process model instance are matched in those of the project plan model. It is described as the following equation:

$$Compliance_{ACT} = \frac{\#\{AN_{actual} \cap AN_{plan}\} + \#\{AF_{actual} \cap AF_{plan}\}}{(\#AN_{actual} + \#AF_{actual})} \quad (6)$$

The degree of activity-oriented deviation( $Deviation_{ACT}$ ) is defined as how many missed and added activity nodes and artifact flows exist in the actual process model instance against the project plan model. It is described as the following equation:

$$Deviation_{ACT} = \frac{W_M Dev_{Miss} + W_A Dev_{Add}}{\max\{W_M, W_A\}} \quad (7)$$

$Deviation_{ACT}$  is formulated with  $Dev_{Miss}$ ,  $Dev_{Add}$ , and two weights, which are  $W_M$  and  $W_A$ . While  $Dev_{Miss}$  describes the deviation with viewpoints of missed activities and artifact flows which were planned to perform but did not,  $Dev_{Add}$  presents the deviation with viewpoints of added activities and artifact flows which were not expected

but were executed.  $W_M$  and  $W_A$  are assigned to each of the  $Dev_{Miss}$  and  $Dev_{Add}$ . We can give a value to each weight according to the relative effect of these deviations to the process performance. For example, the software development processes of a certain company have a tendency to execute without doing the partial activities or merge the some activities to one activity of the project plan. In this case, the deviation with viewpoints of missed activities and artifact flows is the important issue to the company. Therefore, the process engineer of this company can assign the more heavy value to  $W_M$  than  $W_A$ . Like this, any organization can tune the values of two weights according to its important process issue. We do not need to care the ranges of these weights because the regulation of Eq.(7) is conducted by the division with  $\max\{W_M, W_A\}$ . The deviations with aspects of the missed( $Dev_{Miss}$ ) and added( $Dev_{Add}$ ) execution are defined as follows:

$$Dev_{Miss} = \frac{\#\{AN_{plan} - (AN_{actual} \cap AN_{plan})\}}{\#AN_{actual} + \#AF_{actual}} + \frac{\#\{AF_{plan} - (AF_{actual} \cap AF_{plan})\}}{\#AN_{actual} + \#AF_{actual}} \quad (8)$$

$$Dev_{Add} = \frac{\#\{AN_{actual} - (AN_{actual} \cap AN_{plan})\}}{\#AN_{actual} + \#AF_{actual}} + \frac{\#\{AF_{actual} - (AF_{actual} \cap AF_{plan})\}}{\#AN_{actual} + \#AF_{actual}} \quad (9)$$

We can use the  $Compliance_{ACT}$ ,  $Dev_{Miss}$ ,  $Dev_{Add}$ , and  $Deviation_{ACT}$  under various cases. When the comparison between the degree of compliance and deviation of the specific process is needed,  $Compliance_{ACT}$  and  $Dev_{Miss}$  can be used. They represent the portions of the degree of compliance and deviation with actual process aspects. Thus, their sum is 1. Besides, when we want to know about the main causes of process deviation, we can check the values of  $Dev_{Miss}$  and  $Dev_{Add}$  as the first step of finding the causes of the deviation. When the comparisons of the degree of deviation among the past projects are needed, we can use  $Deviation_{ACT}$  which is included two weights.

**Artifact-Oriented Compliance and Deviation.** There are many deviations with the viewpoint of the artifact in the actual process. For example, the artifacts may not be produced as planned to be produced, and the new artifacts can be produced despite of the absence of the planning. In addition, the consideration of the reference relation between artifacts is important because it can give an effect to the quality of the artifact. For instance, the design document and code are influenced by the existence of the requirement specification document. To decide the degree of the compliance and deviation with an artifact-aspect, MPAF provides the  $Compliance_{ART}$  and  $Deviation_{ART}$  which can be defined, similarly to the activity-oriented compliance, using the artifact nodes and arcs according to the definition of AG. The degree of artifact-oriented compliance( $Compliance_{ART}$ ) is defined as how many artifact nodes and reference relations of the actual process model instance are matched in those of the project plan

model. Besides, the degree of activity-oriented deviation( $Deviation_{ART}$ ) is defined as how many missed and added artifact nodes and reference relations exist in the actual process model instance against the project plan model. The compliance and deviation on the artifact nodes are defined as in the following equations:

$$Compliance_{ART} = \frac{\#\{R_{actual} \cap R_{plan}\} + \#\{F_{actual} \cap F_{plan}\}}{\#R_{actual} + \#F_{actual}} \quad (10)$$

$$Deviation_{ART} = \frac{W_R(AFDev_{Miss}) + W_F(AFDev_{Add})}{\max\{W_R, W_F\}} \quad (11)$$

$Deviation_{ART}$  is formulated with  $AFDev_{Miss}$ ,  $AFDev_{Add}$ , and two weights, which are  $W_R$  and  $W_F$ . While  $AFDev_{Miss}$  describes the deviation with viewpoints of missed artifact nodes and reference relations which were planned to be produced but did not,  $AFDev_{Add}$  presents the deviation with viewpoints of added artifacts and reference relations which were not expected but were produced. The meaning and usage of two weights are the same as  $W_M$  and  $W_A$  of Eq.(7). The deviations with aspects of the missed( $AFDev_{Miss}$ ) and added( $AFDev_{Add}$ ) execution are defined as follows:

$$AFDev_{Miss} = \frac{\#\{R_{plan} - (R_{actual} \cap R_{plan})\} + \#\{F_{plan} - (F_{actual} \cap F_{plan})\}}{\#R_{actual} + \#F_{actual}} \quad (12)$$

$$AFDev_{Add} = \frac{\#\{R_{actual} - (R_{actual} \cap R_{plan})\}}{\#R_{actual} + \#F_{actual}} + \frac{\#\{F_{actual} - (F_{actual} \cap F_{plan})\}}{\#R_{actual} + \#F_{actual}} \quad (13)$$

### 4.3 Stability

The structural pattern of the activity and artifact with the stability aspect can be extracted using the method of counting the number of frequency:

- Step 1.** Make a set which contains all activity nodes(Artifact nodes) of all process model instances in the organization except the temporal activity which is created using incomplete information of PTD,
- Step 2.** Make a set which contains all relations(Direct arcs) of activity nodes(Artifact nodes) which are elements of the activity set created at the previous step,
- Step 3.** Count the number of execution frequency of activity nodes(Artifact nodes) or relations.
- Step 4.** Extract the structural pattern with the proper degree of frequency.

In this approach, we consider three items under the behavioral pattern of the process.

- Concurrency tendency of the activity
- Tendency of delayed date of the activity

To catch the tendency of delayed date of the activity, the ratio of delayed date to planned date of activity is used. We can use this tendency for planning the future project, so the more accurate schedules of the projects are acquired.

- Compliance tendency

## 5 Empirical Study with Industrial Data

In this section, we present the empirical study applying MPAF to the 10 completed projects in a financial company of CMM level 2 in Korea.

### 5.1 Project Environment of Study

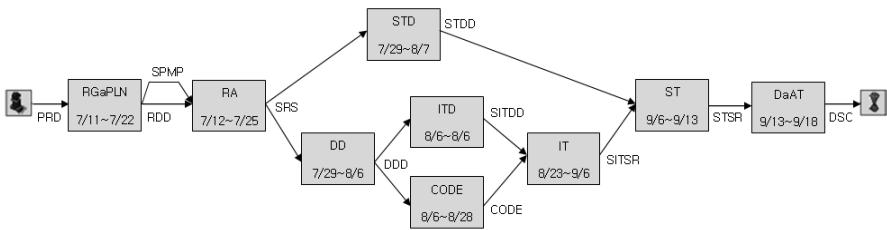
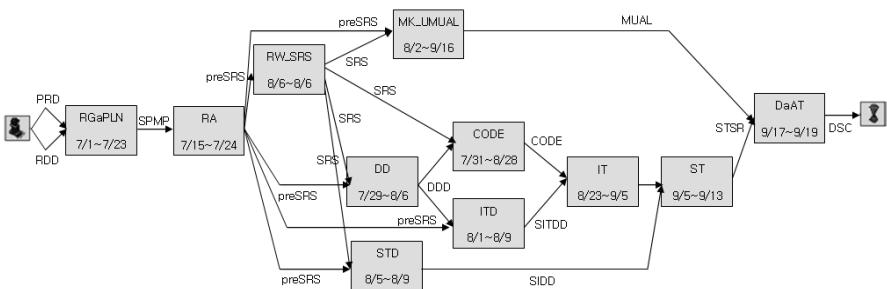
The target projects of this study have been done by the development and maintenance team in a financial company in Korea. They used to develop the small subsystems to extend the legacy system or change the part of functions of the legacy system. Therefore the projects are small scale projects with the average of total days of 50.8 days and the average number of agents involved of 3.7 persons. These projects are executed according to the tailored process while this company has their own development process standard based on the standard waterfall model. To present the extraction of model instance and analysis results of single project, prj-192 is selected among the target projects. The prj-192 observes well the typical pattern of software development works in this company and has the project data which show the various aspects of process execution such as the execution of the unplanned activities, the merge operation of some activities, and so on.

### 5.2 Extraction of Model Instance

According to MPAF, we extract the essential information from PTD to construct the process model. The PTD of this company consists of 31 items such as the measurement of schedule, the measurement of configuration item, and so on. From these items, we choose the schedule-oriented, agent-oriented and artifact-oriented items as the essential information for the process model recovery. Fig.4 shows the project plan model of prj-192 which is constructed at the initial stage of the project and Fig.5 shows the extracted process model instance of the same project using the essential information. We can easily find the differences between the two models described by CogNet.

### 5.3 Analysis Using the Recovered Model Instance

In this section, the results of the process analysis with the analysis factor are represented. Some of the analysis are performed by using the SoftPM.

**Fig. 4.** The project plan model of prj-192.**Fig. 5.** The extracted process model instance of prj-192.**Table 3.** The artifact-oriented performance analysis result of prj-192.

Agent name	Total work days in the conflict state	Frequency of conflicts	Total work days
Jung. K	35	5	119
Jeon. Y	51	6	136
Kim. J	4	2	56

$$AC_{ACTFq} = 4.3 \text{ times } AC_{Time} = 30 \text{ days}$$

**Analysis of Single Project.** The analysis results about the activity-oriented performance are as follows:

- Delayed dates

The most delayed activity is the Detail Design activity(DD). The analysis result is that the activities between the System Test Design activity(STD) and the Integration Test activity(IT), as design and coding phase, are delayed.

- Idle and overlapped dates per activity

The remarkable result is that the following activities after the Requirement Analysis(RA) have the significant idle time. However, DD and the Integration Test Design activity(ITD), and the CODing activity(CODE) are already executed concurrently, although they have clear precedence relation. We can guess that the activities between the design and coding phases are executed concurrently to compensate the idle dates in the previous activities.

- Concurrency of activity

Generally, the concurrency of the design and coding activity is high. This result reflects the same result of analyzing the overlapped dates.

The analysis result of the agent-oriented performance is shown in Table 3. From this result, we can see that the three agents worked two or more jobs concurrently in 30 days during executing prj-192 whose actual duration was 71 days. Besides, from the fact that the average frequency of the agent conflicts is 4.3, prj-192 has the serious agent conflict problems because the actual conflict frequency is 2 times of the planned average frequency of the agent conflicts whose value is 2.3. In a logical sense, if there exists an agent conflict occurred between two activities that have overlapped duration and use the same agent, at least one of the activities must be delayed. While we can predict that the number of delayed activities is 4 or more (because  $AC_{ACTFrq}$  is 4.3), the number of actual delayed activities is 4. Consequently, it shows a positive evidence to our reasoning using  $AC_{ACTFrq}$ .

**Table 4.** The activity-oriented compliance result of prj-192.

	$Compliance_{ACT}$	$Dev_{Miss}$	$Dev_{Add}$	$Deviation_{ACT}$
Value (%)	67.74	0	32.25	32.25

$$(W_M = 1, W_A = 1)$$

**Table 5.** The artifact-oriented compliance result of prj-192.

	$Compliance_{ART}$	$AFDev_{Miss}$	$AFDev_{Add}$	$Deviation_{ART}$
Value (%)	53.85	38.46	46.15	84.61

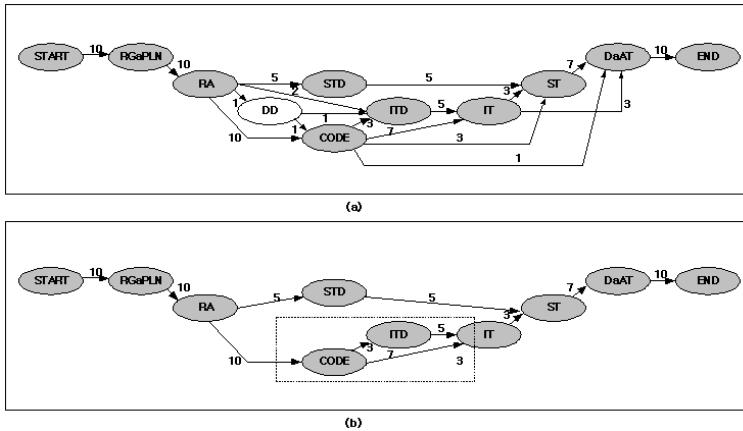
$$(W_R = 1, W_F = 1)$$

The analysis result of activity-oriented compliance with the equations from (6) to (9) are shown in Table 4. This result shows that prj-192 complies well with the project plan. Therefore, we can find that the most of deviations are caused by executing the unexpected activities because  $Dev_{Miss}$  is 0.

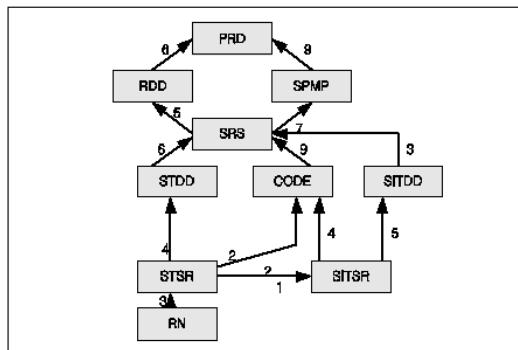
Besides, Table 5 shows the degree of artifact-oriented compliance calculated with the equations from (10) to (13). It shows that prj-192 didn't produce all the artifacts expected in the project plan(38.46%), and it produced the extra-artifacts(46.15%).

**Analysis of Multiple Projects.** Fig. 6 and Fig. 7 show the activity-oriented and artifact-oriented structural pattern, respectively.

In Fig. 6, we can see that the processes executed in this company have the execution tendency shaped as V-model. The remarkable thing in Fig.6(b) is compassed by the dotted rectangle. The DD is absorbed by RA or CODE. Therefore, DD and CODE have the tendency of concurrent execution. The System Integration Test Design Document(SITDD) is produced by CODE though its production is planned at the High-level Design activity(HD). For this reason, we can see that code and SITDD depend on the Software Requirement Specification(SRS) in the Fig.7. According to Fig.7, the target projects have a tendency not to produce the Detailed Design Document(DDD).



**Fig. 6.** The activity-oriented structural pattern: (a)intermediate pattern, (b)final pattern.



**Fig. 7.** The artifact-oriented structural pattern.

The concurrency tendency of activities show that the concurrency is concentrated on the mid-stage of development, such as ITD, CODE, IT, whereas the concurrency of plan is concentrated on the initial stage of development. Due to the absence of the design stage, we can guess that CODE progresses with the design and the test activity.

The tendency of delayed date of the activity has the result that STD, ITD, and IT have higher delay rates than that of the other activities. They are delayed almost two times more than the planned execution dates. On the other hand, the Delivery and Acceptance Test activity(DaAT) has the lower delay rates than that of other activities. This fact is due to the scheduling tendency that the project manager insures the activity execution time. However, the execution time of DaAT is frequently shorter than planned.

The average compliance tendency with the project plan is shown in Table 6. According to Table 6, we can get the result that the processes of the overall projects have the tendency to comply with the project plan from the activity-oriented view. From the artifacts-oriented view, there are some artifacts which are produced at the

unexpected time, and are produced additionally though they are not planned to be produced. The degree of artifact-oriented deviation is relatively high, whereas the degree of the activity-oriented compliance is high. From this fact, we can analogize that this organization may have the problems related to the operation or policy of producing artifacts.

**Table 6.** The average compliance tendency with the project plan.

Category	Measure	Value (%)
Activity-oriented	$Compliance_{ACT}$	94.3
	$Dev_{Miss}$	6.7
	$Dev_{Add}$	38.2
	$Deviation_{ACT}$	44.9
Artifact-oriented	$Compliance_{ART}$	73.5
	$AFDev_{Miss}$	36.5
	$AFDev_{Add}$	44.1
	$Deviation_{ART}$	80.6

$$(W_M, W_A, W_R, W_F = 1)$$

## 6 Related Work

This section introduces related work in reverse engineering in software process area and process validation.

### 6.1 Reverse Engineering in Software Process

The recent approach about reverse engineering in process research area is Cook's approach[9]. Cook et al. has explored methods for automatically deriving a formal model of a process from basic event data collected on the process. The assumption in this approach is that the event stream appears with the recurring pattern of behavior. It terms this form of data analysis "process discovery", because a process is inherent in every project, and for every process there is some model that can be devised to describe it. To develop the technique, this approach transforms the process discovery problem to FSM discovery problem which has been previously investigated. Although this approach suggests the automated event-based framework for discovering the process model, it has the disadvantages such that (1)the result depends on the collected events which are the only fractions of the actual process, (2)the discovered model is very simple because it describes the only activity and activity order except the agent-related and artifact-related information which most of the process model essentially needs, (3)the discovered model dose not provide the structural relations among process elements.

### 6.2 Process Validation

The process validation is related to the check of the actual process compliance with the project plan model. Cugola et al.[10] define a formal framework for checking the

inconsistencies and deviations between the defined process model and the actual process. Their approach is directed the actual process by the process model which is described by LATINE on SENTINEL. It basically defines a policy that tells us when and what data can be considered potentially incorrect by defining the pollution rules. This approach only focuses on finding and resolving the inconsistency and the deviation.

Cook et al.[11] propose the work of process validation trying to measure the compliance using the event-based framework and the string distance metric after the project process is finished. They develop two metric-oriented technique for process validation, from a linear distance measure in terms of event insertions and deletions, to a nonlinear distance measure that takes into account the size of discrepancies. The limitation of this work is that it only provides the metric from the simple activity-related point of view except the consideration of other process elements. Although our compliance metrics are affected by this work, MPAF provides not only the activity-oriented compliance but also the artifact-oriented compliance, and considers the aspects of the artifact flow(the execution order) of activities and the reference relation of artifacts.

## 7 Conclusion and Future Work

In this paper, we have proposed a Model-based Process Analysis Framework(MPAF) for supporting the process analysis from the process element point of view. Since it is not easy to analyze the raw project execution data in the artifacts, a formal process model is used in MPAF for abstracting and understanding of the process. MPAF provides the analysis elements and the procedures which construct the process model instance using information extracted from project execution data, and analyze on the model instance using the defined analysis factors.

The proposed approach has the following advantages.

- It supports to understand and analyze the process with the only concerns that process managers want to know.
- It provides the analysis from the process-oriented viewpoints against the product-oriented viewpoints.
- It can be easily automated.

On the other hand, our approach has a drawback that the analysis factor and the viewpoint depend on the expression power of the process modeling language. However, this drawback can be somewhat offseted by the use of other modeling language.

There are several directions of future work for MPAF:

- To provide the analysis with multi-perspective views of the software process, our research on the measures will be continuously considered with their relations.

- To use as the standard whenever the process compliance is checked, we plan to introduce the quality and maturity model such as CMM to the analysis factor of MPAF.
- To improve the process stability of an organization, the research on the methods of extracting and managing a process pattern will be continuously considered.

We intend to explore these directions, for improving the practicality and usefulness of MPAF.

## Acknowledgement

This work has been supported by the ITRC university support program in Ministry of Information and Communication of Korea.

## References

1. R.S. Pressman . *Software Engineering : A Practitioner's Approach*. Fifth Edition, McGraw-Hill, 2001.
2. S.Y. Min, H.D. Lee and D.H. Bae . SoftPM: a software process management system reconciling formalism with easiness. *Information and Software Technology*, 1-16, 2000.
3. H.D. Lee . *Generating Multiple Perspective from an Integrated Activity-centered Process Model*. M.S. thesis, December 1998.
4. M.C. Paulk et al.. The Capability Maturity Model : Guidelines for Improving the Software Process. Addison-Wesley ,1995.
5. P. Jalote. *CMM in Practice : Processes for Executing Software Projects at Infosys*. Addison-Wesley, 2000.
6. W.A. Florac, and A.D. Carleton. *Measuring the Software Process : Statistical Process Control for Software Process Improvement*. Addison Wesley, 1999.
7. S.W. Ambler. *Process Patterns : Building Large-Scale Systems Using Object Technology*. First Edition, Cambridge University Press, 1998.
8. R.R. Klosch . Reverse Engineering : Why and How to Reverse Engineer Software. *Proceedings of the California Software Symposium(CSS '96)*, 92-99, April 1996.
9. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, Vol 7, No.3, 215-249, July 1998.
10. G.Cugola, E. Di Nitto, C. Ghezzi, and M. Mantione . How to Deal with Deviations During Process Model Enactment. *Proceedings of 17th International Conference Software Engineering*, 265-273, Seattle, Washington 1995.
11. J.E. Cook and A.L. Wolf . Software Process Validation : Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*. Vol 8, No.2, 147-176, April 1999.

# A COM-Based Customization Testing Technique\*

Hojin Yoon, Eunhee Kim, and Byoungju Choi

Department of Computer Science and Engineering  
Ewha Women's University, Seoul, Korea  
{981COG04, ehkim, bjchoi}@ewha.ac.kr

**Abstract.** Component users must customize components they obtain from providers, because providers usually develop components either for general use, or for some other specific purpose. Although the customization is accomplished by modifying the interface of a component, faults caused by customization appear when the implementation part of a component and the interfaces interact. In this paper, we select test cases by inserting faults not into the entire interface but only into specific parts of the interface, which are referred directly by the implementation. They are selected by analyzing the interaction between the interface and the implementation parts. Based on this testing approach, this paper develops a testing technique for a customized COM component. It is applied to a practical component-based system, *Chamois*. Through an empirical study in this paper, it is shown that the specific parts for injecting a fault brings the test cases' effectiveness, which is evaluated.

## 1 Introduction

Modern software systems have become so complicated that software developers are motivated to adopt Component-Based Software Development (CBSD) methodology based on software reuse. CBSD reduces the cost of development by reusing pre-coded software, but there are risks involved in reusing software built by other developers [1]. CBSD consists of Certification, Customization, Composition, and Qualification phases. This paper focuses on the Customization phase. A component is generally released in the form of a black-box, which does not reveal its source code, along with a white-box, also called the *Interface*. Traditional testing techniques cannot be applied to component-based software, because the source code of the components is unavailable [2].

We have proposed a technique for detecting faults that occur through component customization in our earlier papers [3,4]. This technique was based on the general characteristics of the components. Since it is difficult to find the execution path in a component due to the component's black-box, we use the software fault injection technique. It is similar to a mutation test technique, but we do not mutate the entire component. This paper first determines the specific part for mutating, and selects test cases by mutating not the entire component but only specific parts of the component.

---

\* This work was supported by grant number R04-2000-00079-0 from the basic research program of the Korea Science & Engineering Foundation.

This work was partially supported by the Information Technology Research Center.

The specific parts in the white-box are referred directly by the black-box of a component. They make test cases effective, and their effectiveness is evaluated through an empirical study in this paper.

Now, this paper tailors the technique to the Component Object Model (COM) that is one of the most popular component architecture now. With the tailored COM-based technique, it will test *chamois*, which is a component-based system, as an application. Also, an empirical study evaluates the effectiveness of the test cases selected by this technique. The effectiveness of test cases is a particularly important issue, since the purpose of CBSD is to reduce the cost of development. Testing with effective test cases will save testing time.

The remainder of the paper is organized as follows. In Section 2, some related work is discussed. In Section 3, an approach for testing components is proposed and is tailored to COM. In Section 4, it is applied to *Chamois*, and Section 5 analyzes the effectiveness of the proposed technique through an empirical study. In Section 6, this paper presents some concluding remarks.

## 2 Related Research

### 2.1 Component Testing

Voas proposed a method to certify components [5]. Certifying components means assuring that the components do what component users require. Component users must certify components given only a description of what the components do. His certifying method defines five scenarios with three key questions, and uses automated technologies like black-box testing and fault injection to determine whether the component fits each scenario. Once users certify the components, they need to customize the components to fit their requirements.

Harrold proposed a method for the regression testing of component-based software [6]. When new versions of components, that is, customized components, are integrated into an application, the lack of information about such externally developed components makes it difficult to determine the test cases for the application. This regression testing uses metadata provided by component providers. Metadata consists of test frames and test cases covering each test frame. While this method assumes that component providers make the metadata available with the component, we assume that component providers release only the component without any other information.

Interface mutation restricts the operators to model integration faults, and the operators are applied only to function-calls, parameters or global variables that are related to the module interface [7]. Interface mutation was applied to component-based applications [8]. It devised specific operators for component-based applications. This work is closely related to the technique proposed in this paper, since this is for component-based applications and is based on mutation testing. Therefore, an empirical study in Section 5 uses the interface mutation. Interface mutation for component-based applications begins with interface mutation for integration testing. The interface is the connection part of the server component and the client in a component-based application. An empirical study in Section 5 evaluates the effectiveness of these two techniques.

## 2.2 Component Customization Testing

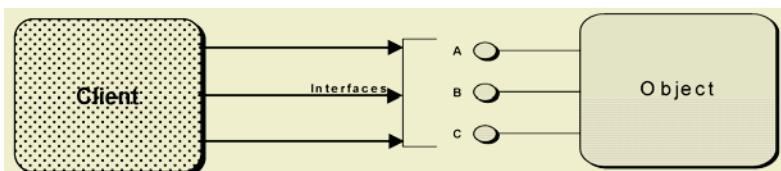
Souza defined component customization [9]. Specifically, the component hides its internal behavior, yet provides an interface whereby the component user can change the component's behavior or its methods. Component customization includes every activity to modify the existing component for reuse. It includes simple modifications to attributes, and adding new domain-specific functions as well as assembling with other components to alter functions. Component architectures, such as the EJB and COM, include all three types of customizations: modification of attributes, addition of domain-specific functions, and assembly of components.

The Component Customization Testing tests the faults occurring in component customization. In the previous work [3,4], the component was defined as a combination of a black-box part and a white-box part, and was tested if a fault resulted from component customization. The black-box part is expected to be error-free and the white-box part can be tested with other established techniques because it is open. Thus, it could be said that there are no faults in either the black-box part or the white-box part. Even though each black-box part and white-box part is error-free, there is a possibility of a critical fault when they are integrated to form a customized component. This kind of fault occurs in the interaction between the black-box part and the white-box part, and can be detected by integration testing. Consequently, component customization testing is a form of integration testing of the black-box part and the white-box part of a customized component.

## 2.3 Component Object Model

The Component Object Model (COM) is a software architecture that allows applications to be built from binary software components. COM is a platform-independent, distributed, object-oriented system for creating binary software components that can interact. COM is convenient to adopt a standard pictorial representation for objects and their interfaces. An object is an instantiation of some class. The interface has a specific name or type and names of member functions; it defines only how one would use that interface and what behavior is expected from an object through that interface [10].

The interfaces extend toward the client, and the client is understood to have a pointer to one or more of the interfaces on that object as illustrated in Figure 1. In Figure 1, the interface is White-box, and the object is Black-box.



**Fig. 1.** Interaction between COM component and client

### 3 A Testing for Customized Components

As mentioned in Section 2, component customization testing is a form of integration testing of the two parts of a component, and the two parts were defined as a Black-box class and a White-box class [3,4]. The testing proposed in this paper first defines which part of a component could be a Black-box class and which part of a component could be a White-box class, and then selects some specific parts where faults are injected. With the specific parts, it makes fault-injected versions of a customized component in order to select test cases that differentiate fault-injected versions from a customized version.

#### 3.1 Definitions

**Definition 1. Black-box class ( $B$ )**

The Black-box class is a part of the component where the source code is not disclosed and therefore cannot be modified for customization. This is denoted as  $B$ . Usually the component provider makes  $B$ .

**Definition 2. White-box class ( $W$ )**

The White-box class is a part of the component where the source code is open to the user and can be modified for customization. The white-box class is denoted as  $W$ . Usually the component provider creates  $W$ , and the component user modifies  $W$  for customization. The customized interface is denoted as  $cW$  and the fault-injected into  $cW$  is denoted as  $fW$ .

**Definition 3. Component ( $BW$ )**

The component is a combination of  $B$  and  $W$ , and so is denoted as  $BW$ . The  $W$  transformed through the component customization to  $BW$  is called  $cBW$ , and the  $cBW$  with the fault injection is called  $fBW$ .

Since it is difficult to find the execution path in a component due to the component's  $B$ , we use the software fault injection technique. The key factor in the software fault injection technique is to determine where to inject the fault. Of course, the test cases that are selected by injecting a fault into the entire  $cW$  or the 'customization code', which is added or modified in  $W$  for component customization, may detect errors in component customization. However, the effectiveness of a test case is an important issue in CBSD. Thus, we need to select the specific part where a fault is injected. The specific part is based on the interaction between  $B$  and  $W$ .  $B$  communicates with  $W$  whenever a component is executed. All elements of  $W$  don't interact directly with  $B$ . The elements in  $W$  communicating directly with  $B$  are defined as the specific part which we insert fault into. The part is not the place customization faults exist, and not even the place modified through customization.

**Definition 4. Fault Injection Target ( $FIT$ )**

This is the specific part into which a fault is injected. More effective test cases can be selected by injecting a fault only into the  $FIT$ .  $FIT$  is defined as  $DRE$ .

**Definition 5. Fault Injection Operator ( $FIO$ )**

$FIO$  is an operator that defines how to inject a fault that will not cause syntax errors.

We inserted a more sophisticated approach into our technique, which is the customization pattern. We don't apply the same *FIT* and *FIO* to all of the customized components. We first identify which customization pattern was used, and then the *FIT* and *FIO* only for that customization pattern are applied in order to get more effective test cases. The customization patterns should be categorized first before *FIT* and *FIO* are defined. We proposed how to categorize customization patterns in our earlier paper [3]. It has two dimensions; one is syntactic pattern, and the other is semantic pattern. The syntactic pattern decides the syntactic way of customization and it is expressed as *Pattern.syn*. The semantic pattern decides the semantic way of customization and it is *Pattern.sem* for short. There are two patterns in *Pattern.syn*, and three patterns in *Pattern.sem* as shown in Table 1.

**Table 1.** *Pattern.syn* and *Pattern.sem*

Pattern	Description
<i>Pattern.syn</i> <sub>1</sub>	A pattern for injecting a customization code directly into <i>W</i>
<i>Pattern.syn</i> <sub>2</sub>	A pattern that generates <i>New</i> , which has the customization code, and allows <i>W</i> to reference the <i>New</i> by setting the association between the two.
<i>Pattern.syn</i> <sub>3</sub>	A pattern that generates <i>New</i> , which has the customization code, and allows <i>W</i> to reference by setting the inheritance between the two.
<i>Pattern.sem</i> <sub>1</sub>	A pattern that creates <i>cBW</i> by modifying <i>W</i> to set the component properties.
<i>Pattern.sem</i> <sub>2</sub>	A pattern with another component or <i>New</i> which adds or changes the component's behavior according to the user's development requirements.

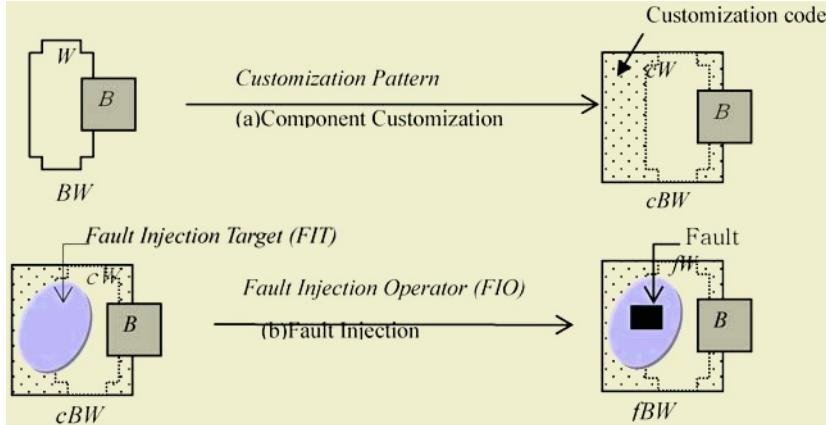
Figure 2 summarizes the approach to test customized components. The dotted part represents the customization code. For testing the customized component *cBW* in Figure 2(a), we create a *fBW* by injecting a fault into the *FIT* of *cW* as shown in Figure 2(b). We call the fault injection operator *FIO*. There are several patterns in turning *BW* into *cBW* through component customization as shown in Figure 2(a). The targets and the operations of injecting a fault into *cBW* vary according to these patterns. Therefore, we first categorize the customization patterns, and then we decide where the *FIT* should be in *cBW* for each customization pattern.

The component-based software test data are sequences of methods. It selects the method sequence format, which can differentiate the test target, *cBW* from the *fBW* generated by applying *FIO* in accordance with the component customization pattern. The component-based software test data selection technique is directly used from the definitions as follows:

**Definition 6.** Test case selection for component customization:

A test case for component customization is a sequence of method calls that differentiate *fBW* from *cBW*. *TC* is a set of these test cases.

$$TC = \{ x \mid cBW(x) \neq fBW(x), \text{ where } x \text{ is a method sequence.} \}$$



**Fig. 2.** Component customization and fault injection

Definition 6 looks similar to the mutation test case selection technique [11], but it does not select the test cases with the mutants made by mutating all the elements of  $cBW$  as mutation testing does. It uses the mutants made by mutating only the  $FIT$ , based on the interaction between  $B$  and  $cW$ .

### 3.2 A Testing for Customized COM Components

Now, this section tailors the technique this paper proposed in Section 3.1 to COM component architecture. A COM component consists of an object and interfaces. An object is made of classes that have main functions of a component. It could be  $B$  since it doesn't open its source code and it has main code of its functions. An interface of a COM component is accessible from outside, so it is  $W$  in COM. Customization patterns of COM should be defined first before selecting  $FIT$  and  $FIO$  in COM.

**Table 2.** Customization Patterns in COM

Pattern	Definition
Pattern 1	A pattern which sets the values of properties by changing interface
Pattern 2	A pattern which sets the values of properties by making new interface.
Pattern 3	A pattern which adds new functions by changing interface
Pattern 4	A pattern which adds new functions by making new interface
Pattern 5	A pattern which modifies functions by inserting new sequence to an interface
Pattern 6	A pattern which modifies functions by making new interface and inserting new sequence into the new interface

The Patterns in COM match *Pattern.syn* and *Pattern.sem* as shown in Table 3.

Once the customization patterns in COM are categorized,  $FIT$  and  $FIO$  in every pattern should be defined according to Definition 4 and Definition 5. As mentioned in Section 3.1,  $FIT$  is a specific part of  $W$  which communicates with  $B$  directly. It's not a modified part through customization. In order to select  $FIT$  of COM component,

many practical COM components should be considered, and COM components coded with VB are analyzed in this paper. *FITs* in every pattern are decided by analyzing VB COM components as shown in Table 4.

**Table 3.** Customization Patterns of COM vs. *Pattern.syn* and *Pattern.sem*

<i>Pattern.syn</i> <i>Pattern.sem</i>	<i>Pattern.syn<sub>1</sub></i>	<i>Pattern.syn<sub>2</sub></i>	<i>Pattern.syn<sub>3</sub></i>
<i>Pattern.sem<sub>1</sub></i>	1	2	
<i>Pattern.sem<sub>2</sub></i>	3, 5	4, 6	

**Table 4.** *FITs* and *FIOs* in VB COM

Patterns	<i>FITs</i>	<i>FIOs</i>
Pattern 1	<i>Dim</i> : A property declared with Dim	<i>MDim</i> : Modify the value of the property declared with “Dim”
	<i>Form</i> : A property declared with Form	<i>MForm</i> : Modify the value of the property of a “Form”
Pattern 2	<i>DimN</i> : A property declared with Dim in a new Interface	<i>MDimN</i> : Modify the value of the property declared with “Dim” in a new Interface
	<i>FormN</i> : A property declared with Form in a new Interface	<i>MFormN</i> : Modify the value of the property of a “Form” in a new Interface
Pattern 3	<i>PublicSub</i> : A function declared with <i>Public Sub</i>	<i>RSub</i> : Replace a function with a new component <i>CSub</i> : Create a new function
	<i>PublicFunction</i> : A function declared with <i>Public Function</i> .	<i>RFnt</i> : Replace a function with a new component <i>MPrm</i> : Modify the value of the parameter of a new function
Pattern 4	<i>PublicSubN</i> : A function declared with <i>Public Sub</i> in a new Interface	<i>RSubN</i> : Replace a function with a new component in a new Interface <i>CSubN</i> : Create a new function in a new Interface
	<i>PublicFunctionN</i> : A function declared with <i>Public Function</i> in a new Interface	<i>RFntN</i> : Replace a function with a new component in a new Interface <i>MPrmN</i> : Modify the value of the parameter of a new Function in a new Interface
Pattern 5	<i>PublicSub</i> : A function declared with <i>Public Sub</i>	<i>CSubS</i> : Create a new function including a new sequence
	<i>PublicFunction</i> : A function declared with <i>Public Function</i> .	<i>MPrmS</i> : Modify the value of the parameter of a new function including a new sequence
Pattern 6	<i>PublicSubN</i> : A function declared with <i>Public Sub</i> in a new Interface	<i>CSubSN</i> : Create a new function including a new sequence in a new Interface
	<i>PublicFunctionN</i> : A function declared with <i>Public Function</i> in a new Interface	<i>MPrmSN</i> : Modify the value of the parameter of a new Function including a new sequence in a new Interface

## 4 An Application to *Chamois*

This section describes how to apply the technique we developed in Section 3 with a component-based system, *Chamois* [12].

### 4.1 Chamois

*Chamois* is a component-based system developed with COM and Java Web Service. It supports common APIs for commercial software and in-house components. Figure 3 shows the architecture of *Chamois*.

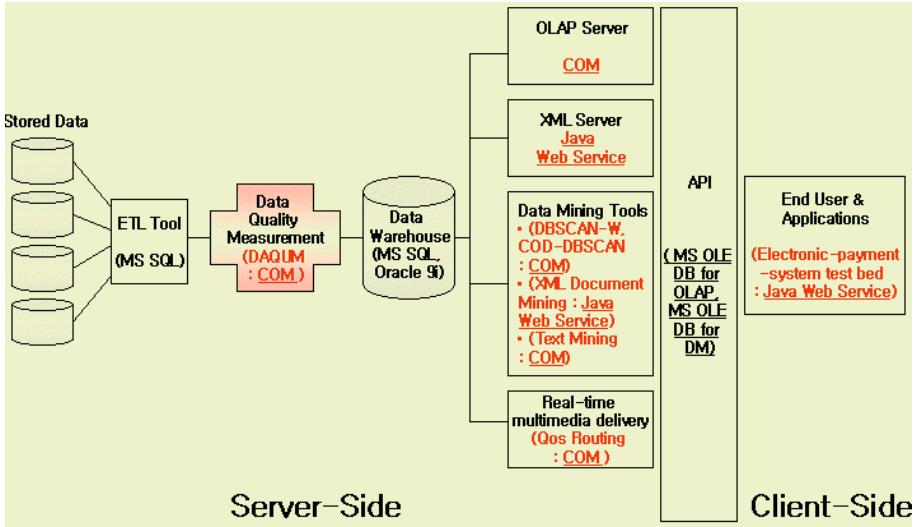
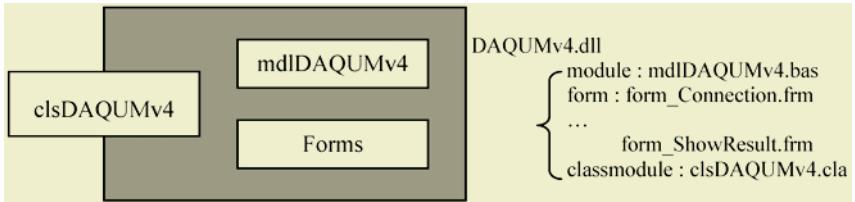


Fig. 3. *Chamois* Component Systems

*Chamois* consists of various commercial components and in-house components under IKEA (Integrated Knowledge Engineering Architecture) as follows:

- Commercial Components: ETL Tool (MS SQL), Data Warehouse (MS SQL, Oracle 9), API (MS OLE DB for OLAP, MS OLE DB for DM)
- In-house COM Components: DAQUM, OLAP Server, DBSCAN-W, COD-DBSCAN, Text Mining, Real-time multimedia delivery: Qos Routing
- In-house Java Web Service Components: XML Server, XML Document Mining, Electronic-payment-system test bed

This section picked up DAQUM component, which is newly developed for *Chamois* as an in-house COM component. DAQUM component measures and controls the data quality based on dirty data taxonomy [13]. Figure 4 shows DAQUM component. In Figure 4, *clsDAQUMv4* is *W* as a class module of DAQUM, and a unit of *mdlDAQUMv4* and *Forms* is *B*, according to Definition 1 and Definition 2. From now on, the technique proposed in Section 3.2 will test DAQUM with its *B* and *W*.

**Fig. 4.** Structural elements of the DAQUM component

<pre> Dim Constr As String Dim      setDSN   As     New DSNConnection.ClsDSNCon  Public Sub DASStart()     NewConnection     SetProperty     DirtyDataClean x, y End Sub  Public Sub Connection()     Form_Connection.Show     vbModal     Constr           =     Form_Connection.Constr End Sub  <b>Public Sub NewConnection()</b>     Set      setDSN   = new     DSNConnection.ClsDSNCon     setDSN.DSNConnection     Constr = setDSN.Constr End Sub </pre>	<pre> Dim Constr As String Dim      setDSN   As     New DSNConnection.ClsDSNCon  Public Sub DASStart()     Connection     SetProperty     DirtyDataClean x, y End Sub  Public Sub Connection()     Form_Connection.Show     vbModal     Constr           =     Form_Connection.Constr End Sub  Public Sub NewConnection()     Set      setDSN   = new     DSNConnection.ClsDSNCon     setDSN.DSNConnection     Constr = setDSN.Constr End Sub </pre>
(a) <i>cBW</i>	(b) <i>fBW</i>

**Fig. 5.** *cBW* and *fBW* of DAQUM

## 4.2 A Customization

Actually, DAQUM can be customized in every customization pattern we defined in Table 2. However, in this section, we customize it only in COM customization pattern 3 as an example. This section shows how the technique of Section 3.2 tests a customized component. It is customized to add a new function that checks if it is connected to DSN or not by changing a connection to DSNConnection of Figure 5 (a). This customization code is expressed with bold font in Figure 5 (a). Unfortunately, DSNConnection has a fault on Connection Value, *Constr*. This section tests DAQUM component in order to detect the fault DSNConnection has through customization. The technique in Section 3.2 will be used to test it.

## 4.3 Test Case Selection

To select test cases, *fBW* is made first by applying *FITs* and *FIOs* of Table 4. To generate *fBW*, the customization pattern of *cBW* should be decided first. In this case, *FIT*,

*PublicSub* and *FIO*, *RSub* were picked up since *cBW* in Figure 5 (a) is customized with Sub of the COM customization pattern 3. If a *cBW* is customized in more than one customization pattern, more than one *FIT* and *FIO* would be picked up, and more than one *fBW* would be generated. We picked up one *FIT* and one *FIO* in this case since *cBW* in Figure 5 was only in the customization pattern 3. Once *FIT* and *FIO* are selected, they are applied to *cBW*. *PublicSub* in *cBW* could be *NewConnection* of *DAStar()* and *RSub* replaces it with *Connection()*, in order to generate *fBW*. We have finally made one *fBW* of Figure 5 (b) with one *FIT* and one *FIO*.

Once *fBW* is generated with *PublicSub* and *RSub*, the test cases can be selected according to Definition 6. Table 5 shows a part of the test cases that differentiates the result of *cBW* and that of *fBW*. Every test case has its own expected output, as shown in Table 5. With the test cases selected by our approach, we could test whether *cBW* has a fault or not. If the result of *cBW* with a test case is different with the expected output of the test case, it could be said that *cBW* has a fault.

**Table 5.** A part of the test cases for testing customized DAQUM component

Test data	<i>cBW(x)</i>	<i>fBW(x)</i>
DSN: “Chamois”, ID: “sa”, PWD: “1234”	No DSN connection value	Form_DirtyDataClean Emp, emp
DSN: “Chamois”, ID: “sa”, PWD: “ ”	No DSN connection value	Form_DirtyDataClean Emp, emp
DSN: “Chamois”, ID: “ ”, PWD: “1234 ”	Connection failed	No DSN connection value
DSN: “ ”, ID: “sa”, PWD: “1234”	Connection failed	No DSN connection value
DSN: “Chamois”, ID: “ ”, PWD: “ ”	Connection failed	No DSN connection value
...	...	...

#### 4.4 Testing

Among the test cases in Table 5, a test case, “DSN: “Chamois”, ID: “sa”, PWD: “1234”” is selected and applied to *cBW*. The component user expects the *cBW* for this test case to result in “Form\_DirtyDataClean Emp, emp”. However, applying “DSN: “Chamois”, ID: “sa”, PWD: “1234”” to *cBW* results in “No DSN connection value”. It shows that this *cBW* has a fault.

## 5 An Empirical Study

A set of test cases with high effectiveness and a small number of mutants or *fBWs* could reduce the testing cost by reducing testing time, which is the purpose of CBSD. This section will do an empirical study with the technique tailored to COM components, as described in Section 3.2.

## 5.1 Effectiveness Criteria

The effectiveness of test cases in this section is evaluated in terms of two criteria;  $Eff_1$ , and  $Eff_2$ ,

$$Eff_1 = (\text{number of faults detected by the set of test cases} / \text{total number of faults}) * 100$$

$$Eff_2 = (\text{number of test cases with fault detection} / \text{total number of test cases}) * 100.$$

$Eff_1$  measures how many faults are detected for the set of test cases [14].  $Eff_2$  measures how many test cases detect faults out of the selected test cases [15]. The empirical study in this section evaluates  $Eff_1$  and  $Eff_2$ . Four results will be analyzed to demonstrate the merit of the technique:  $Eff_1$ ,  $Eff_2$ , the number of fBWs, and  $Eff_2$  per fBW.

This empirical study compared the technique of this paper with the interface mutation for the following three reasons: First, the technique tests customized components. Although a few papers have proposed methods to test components, there is no well-known approach to test customized components. Second, the technique proposed in this paper uses the interface of a component to test a component. The interface mutation also uses the interface of a component. The difference lies in which part of the interface is used for testing. The technique of this paper used the *FIT* of the interface, and the interface mutation defined its counterpart for the *FIT*. Third, the testing technique in this paper is detailed enough to be applied to a real component. It was tailored to COM components in Section 3. An empirical study needs detailed resultant values, but some research that proposes approaches to test components are not that detailed. The interface mutation is sufficiently detailed to generate real test cases. Therefore, the interface mutation can be an appropriate counterpart in the empirical study.

The interface mutation was applied to the CORBA distributed-object-architecture, while some black-box testing or white-box testing techniques are easily applied to normal stand-alone programs that generally input and output data. The component architecture is similar to a distributed-object-architecture since both have interface and implementation parts for a component. A test case of the technique proposed in this paper or the interface mutation is a form of method-calls implemented in a server. Therefore, the testing technique proposed in this paper is compared with the interface mutation rather than black-box testing or white-box testing.

## 5.2 Using the Interface Mutation for Component-Based Applications

Interface mutation generates mutants by injecting faults into some elements within the interfaces of the components. Operators for interface mutation were suggested for CORBA IDL, but not for COM, as follows [8]:

1. *Replace*: Operator for replacing an occurrence of one of “in”, “out” and “inout” with another in the list.
2. *Swap*: Operator for parameter swapping in the method call. Parameters of the same type could be swapped in a method call.
3. *Twiddle*: Operator for replacing a variable  $x$  with  $\text{succ}(x)$  or  $\text{pred}(x)$ .

4. *Set*: Operator for assigning a certain fixed value to a parameter or to the value returned from a method.
5. *Nullify* : Operator for nullifying an object reference.

CORBA IDL is similar to the interface of COM. All operators above can be applied to VB COM. The Interface in COM is equivalent to the interface in interface mutation, since either of them has method-signatures defined in a server program. This empirical study evaluates the effectiveness of test cases selected by interface mutation through the following procedure.

- Make a list of methods signatures from the Remote Interface.
- Generate mutants using the interface mutation operators.
  - Select test cases from mutants.
  - Apply test cases to an EJB application.
  - Check whether each test case detects a fault or not.
  - Evaluate the effectiveness with the number of test cases that detected a fault.

According to interface mutation, how many and which operators to use depends on the resources available for testing and the criticality of the application domain. The empirical study generates one mutant using one operator, and counts the number of mutants, which can be killed, in order to get as high an effectiveness as possible.

### 5.3 Using the Technique of This Paper

In this empirical study, the technique is applied to COM components through the following procedure.

- Identify the customization pattern by analyzing a customized COM component.
- Generate *fBWs* with *FIOs* of the customization pattern.
  - Select test cases from *fBWs*.
  - Apply test cases to the COM component.
  - Check whether each test case detects a fault.
  - Evaluate the effectiveness with the number of test cases that detected a fault.

### 5.4 COM Components

The empirical study in this section has picked up the following five COM components as shown in Table 6. They each have their own customized versions, and the testing technique is applied to their customized versions. For evaluating effectiveness, it is assumed that customizations of the five components in Table 6 cause faults. This means the customized-versions of the five applications in Table 6 contain faults, and the faults will be detected through this empirical study.

### 5.5 Results and Analysis

Every COM component in Table 6 was tested by both the technique of Section 3.2 and the interface mutation with 20 test cases. Table 7 shows the values that came through the testing of them. IM in Table 7 means the interface mutation, and FIT/FIO

**Table 6.** COM components in the empirical study

COM components	Pattern	Description
AppStack	Pattern4	It pushes and pops data that a user inputs
AppOperator	Pattern1	It calculates two values with an operator. The user inputs two values and an operator.
AppBooking	Pattern6	It shows tables in a restaurant, and the user picks one. It generates a reservation number, and saves it on DB.
AppZipcode	Pattern3	It searches a zip code with an address the user inputs.
AppVoting	Pattern5	The user selects an example of a question. It shows how many users selected the item so far.

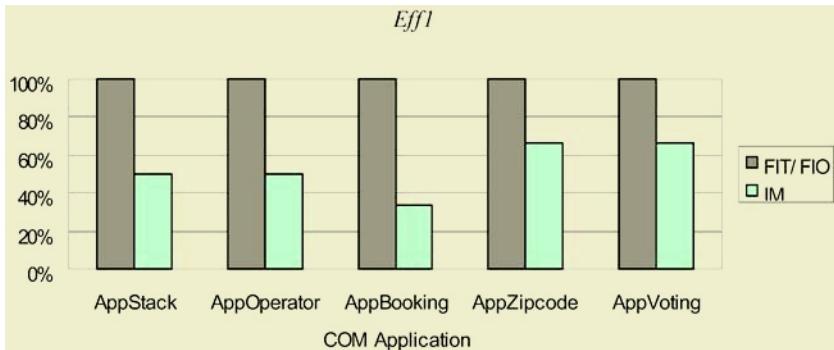
**Table 7.** The Values from the Empirical Study

Components		AppStack	AppOperator	AppBooking	AppZipcode	AppVoting
$Eff_1$	number of faults	2	2	3	3	2
	number of faults detected by FIT/FIO	2	2	3	3	2
	number of faults detected by IM	1	1	2	2	2
$Eff_2$	number of fBWs	2	1	1	2	1
	number of mutants	1	4	2	2	6
	number of test cases detecting faults in IM	10	10	6	18	20
	number of test cases detecting faults in FIT/FIO	17	20	16	20	20

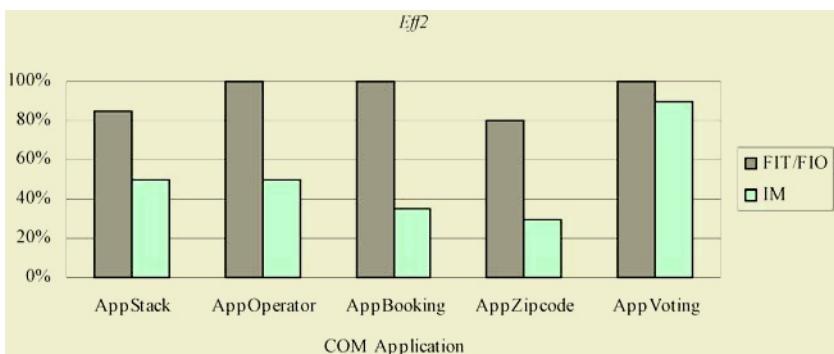
means the technique proposed in this paper. First three rows in Table 6 are for  $Eff_1$ , and last four rows are for  $Eff_2$ .

Most of  $Eff$ 's are 100% in the empirical study as seen in Figure 6. FIT/FIO generates fBWs on the customization patterns of an application. It means that each fBW covers each customization pattern. That is because customization patterns are related to their own customized-parts and faults are through the customized-parts. Therefore, once an fBW is generated with a customization pattern, test cases that differentiate the fBW from the cBW tend to cover faults of the customized-part of the pattern.

FIT/FIO shows greater  $Eff_2$  than interface mutation in Figure 7. The first reason is the way in which customization patterns are identified. While the interface mutation generates mutants by applying every operator to every method-signature defined in the interface, FIT/FIO generates fBWs by applying some operators selected with the customization pattern. Test cases selected with a customization pattern can be more sensitive to the fault than those without identifying customization patterns, because operators in each customization pattern are devised by considering faults that can occur in the pattern.



**Fig. 6.**  $Eff_1$  of five COM applications.

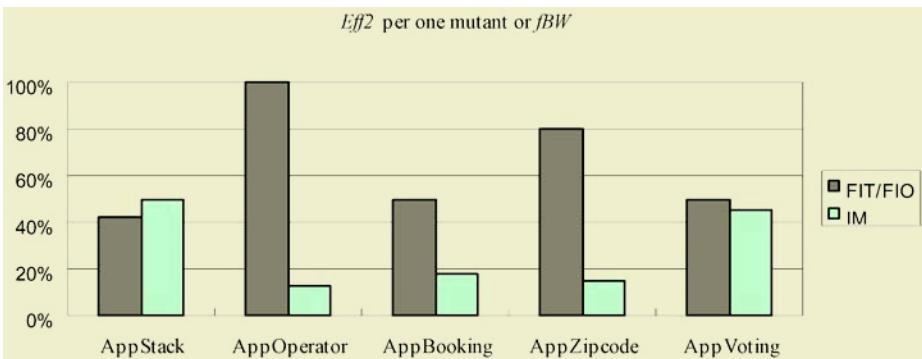


**Fig. 7.**  $Eff_2$  of FIT/FIO and IM

Figures 7 show that FIT/FIO has greater effectiveness. If a technique selects effective test cases but needs many mutants or *fBWs*, it is hard to say it is effective, because the cost for making and handling mutants or *fBWs* may lower the effectiveness. That is why the empirical study evaluates the effectiveness per *fBW* or mutant. That's why the effectiveness per one *fBW* or one mutant is evaluated in Figure 8. In the almost every application, FIT/FIO is better than IM.

The empirical study in this section is not intended to suggest that FIT/FIO is better than interface mutation but to show that FIT/FIO is good for component architecture since it starts with a concept of the component that consists of a black-box part and a white-box part, and has customization patterns, *FITs*, and *FIOs*. Finally, the empirical study shows that the test cases selected from an *fBW* made with *FITs* and *FIOs* can be sensitive to faults that are liable to occur in a customization pattern, since *FITs* were defined for each customization pattern.

If the mutation test is simply applied to the interface of a component, every element of the interface would be mutated, and many mutants would be generated for testing. Using AppOperator in Table 6 as an example, the mutation testing generates 192 mutants by mutating every statement, variable, and operator in the interface of AppOperator. With these 192 mutants, the mutation testing selects test cases that kill them. They are approximately 288. They could detect all the faults AppOperator has, and  $Eff_1$  could be 100%. But all the test cases do not detect the faults, so  $Eff_2$  is evalu-



**Fig. 8.**  $\text{Eff}_2$  for each Mutant of IM and FIT/FIO

ated as 62.5% in this case. Compared to this, FIT/FIO generates only 4  $fBW$ s for testing AppOperator in this empirical study. Nevertheless, it has better  $\text{Eff}_2$  than the mutation testing. Therefore, it could be said that the technique proposed in this paper has a fault detecting ability on test cases, while it solves the computational bottleneck, which is caused by so many operators and mutants.

## 6 Concluding Remarks

The lack of the source code of components makes it difficult for component users to test components. This paper has tailored the component customization testing technique [3,4] to COM components. It has also evaluated the effectiveness of the test cases with COM components. This paper is significant for the following three reasons:

First, it enables component users to test faults occurring in interactions between the implementation part, which is closed, and the interface part, which is available to the component user, although component users can access only the source code of the interface of the components. This paper uses software fault injection techniques to inject a fault into the interface, in order to trigger execution paths without information related to the implementation part.

Second, it is applicable to ‘real’ component architectures. The specific *FITs* and *FIOs* were developed for COM in Section 3, and an example showed how the *FITs* and the *FIOs* are applied. It is possible to apply the technique of this paper to various component architectures, because the definitions of the *B*, *W*, *FIT*, and *FIO* came from the component’s general characteristics and can be tailored to any specific component architecture as shown on COM in this paper and EJB[16]. Section 4 shows how to apply the technique tailored to the COM component by testing *Chamois* component-based system.

Third, it selects effective test cases. The empirical study evaluated the effectiveness with the two definitions of the effectiveness:  $\text{Eff}_1$  and  $\text{Eff}_2$ . The results of the empirical study were showed in Section 5. The effectiveness is one of the important requirements in CBSD. The effective test cases can save testing time, consistent with the purpose of component-based software, namely, a major reduction in the cost of

software development. The empirical study showed that the *FIT* generates a reasonable number of *fBWs*, selecting effective test cases, which leads to increased effectiveness.

The technique of this paper can be applied to other component architectures by tailoring the definitions of *B*, *W*, *FIT*, and *FIO*. We have already done an empirical study on EJB component in an earlier paper [16]. It showed that this technique applied to EJB components leads the effective test cases in terms of *Eff1* and *Eff2*. Finally, this paper has developed a testing technique for COM components, and the empirical study in this paper proves that this technique applied to COM components also contributes to greater effectiveness. The testing techniques of this paper will work well on component-based development as shown in the empirical study with COM and that with EJB.

A tool has been built for EJB components. It tests EJB components automatically using the technique tailored to EJB, and it was used in the empirical study with EJB [16]. A tool for COM components can be built now. Moreover, the approach in this paper will be applied to test web-based software engineering, which seems to be very popular these days.

## References

1. Weyuker.E.J.: Testing Component-Based Software: A Cautionary Tale. *IEEE Software*, Sep/Oct. (1998) 54-59
2. Mary Jean Harrold, Donglin Liang, and Saurabh Sinha: An Approach To Analyzing and Testing Component-Based Systems. Proceeding of the First International ICSE Workshop on Testing Distributed Component-Based Systems, May. Los Angeles, CA. (1999)
3. Hojin Yoon and Byoungju Choi: Component Customization Testing Technique Using Fault Injection Technique and Mutation Test Criteria. Proceeding of Mutation 2000, Oct. San Jose, USA.(2000) 71-78
4. Hojin Yoon and Byoungju Choi: Inter-class Test Technique between Black-box-class and White-box-class for Component Customization Failures. Proceeding of APSEC'99. Taka-matsu, Japan. (1999) 162-165
5. Jeffrey M. Voas: Certifying Off-the-Shelf Software Components. *IEEE Computer*, Vol. 31, No.6. (1998) 53-59
6. Mary Jean Harrold, Alessandro Orso, David Rosenblum, and Gregg Rothermel: Using Component Metadata to Support the Regression Testing of Component-Based Software. Technical Report GIT-CC-01-38, College of Computing, Georgia Institute of Technology. (2001)
7. Marcio E. Delamaro, Jose C. Maldonado, and Aditya P. Mathur: Interface Mutation: An Approach for Integration Testing. *IEEE Transactions on Software Engineering*, Vol. 27, Vo. 3. (2001) 228-247
8. S. Ghosh: Testing Component-Based Distributed Applications. Ph. D Dissertation, Department of Computer Science in Purdue University. (2000)
9. Desmon F.D'Souza and A.C.Wills: Object, Components, and Frameworks with UML. Addison-Wesley. (1998)
10. Component Object Model, <http://msdn.microsoft.com/library/>
11. R.A.DeMillo, R.J.Lipton, and F.G.Sayward: Hints on Test Data Selection: Help for the Practicing Programmer. *IEEE Computer*, Vol. 11, No. 4. (1978) 34-41
12. Won Kim, Ki-Joon Chae,Dong-Sub Cho, Byoungju Choi, Anmo Jeong, Myung Kim, KiHo Lee, Meejeong Lee, Sang-Ho Lee, Seung-Soo Park, Hwan-Seung Yong: The Chamois Component-based Knowledge Engineering Framework. *IEEE Computer Journal*, May. (2002)

13. Won Kim, Byoungju Choi, Eui-Kyeong Hong, Soo-Kyung Kim, Doheon Lee: A Taxonomy of Dirty Data. *The Data Mining and Knowledge Discovery Journal*, Vol. 7, No. 1. (2003) 81-99
14. W. Eric Wong, Joseph R. Horgan, Aditya P. Mathur, and Alberto Pasquini: Test Set Size Minimization and Fault Detection Effectiveness: A Case Study in a Space Application. *Proceeding of COMPSAC'97*. Washington D.C., USA, (1997) 522-529
15. Aditya P. Mathur and W. Eric Wong: Comparing the Fault Detection Effectiveness of Mutation and Data Flow Testing: An Empirical Study. SERC-TR-146-P, Purdue University. (1993)
16. Hoijin Yoon, Byoungju Choi: Effective Test Case Selection for Component Customization and Its Application to EJB. *The Software Testing, Verification, and Reliability Journal*, to appear on Vol14 No2 June 2004.

# Design Defect Trigger for Software Process Improvement

EunSer Lee<sup>1</sup>, Kyung Whan Lee<sup>1</sup>, and Keun Lee<sup>2</sup>

<sup>1</sup> School of Computer Science and Engineering, Chung-Ang University  
`{eslee, kwlee}@object.cau.ac.kr`

<sup>2</sup> Department of Computer Science, School of Engineering, University of Southern California  
`keunlee@usc.edu`

**Abstract.** This research is intended to develop the empirical relationship between defects and their causes to estimate. Also, using defect cause, we understand associated relation between defects and design defect trigger. So when we archive resemblant project, we can forecast defect and prepare to solve defect by using defect trigger.

**Keywords:** Defect, Defect Trigger, GQM method, Defect cause prioritization, COQUALMO, Defect reduction

## 1 Introduction

The most important goal of software engineering is to develop products of high quality. To achieve this goal, the software engineer must apply efficient methods with cutting-edge tools throughout the software process. Moreover, the software engineer needs to confirm whether a quality product has been produced [1].

A computer system break-down is more frequently caused by the software than by the hardware [1]. Therefore, in order to ensure a quality software product, it is important to develop a highly reliable software by using software engineering technology. In this context, quality has become a vital factor in software engineering.

The quality of the system, application, and the product was used to describe the requirement analysis, design, coding and test stages. An excellent software engineer evaluates the test case developed through analysis, design, source codes, and their documentation. To achieve the required quality level, the engineer must use technical measures to evaluate the quality, which relies on objective methods rather than subjective methods [1][2].

In the context of a project, quality can be explained as following. A project manager must estimate quality while a project is in procedure. The undisclosed matrixes collected individually by a software engineer are integrated in order to determine the project level. Although it is possible to measure various quality elements with merely the collected data, it is also required to integrate the data in order to determine the project level [4].

The key task during the project is to measure errors and defects, even though many quality measurements can be collected. The matrix drawn from these measurements

offers the standard of the effectiveness of quality warranty and the control activity of individual and group software[5].

Errors that are found during examination and testing offer an insight on the efficiency of each activity of the matrix. Furthermore, the error data can be used in calculating the DRE(Defect Removal Efficiency) of each process framework activity. In developing a reliable software, a key factor is to find and remove defects that exist during software development. We infer quality from such factors, and this quality is a main factor that determines the success of a project together with cost and time schedule. The concept of software quality cannot be defined easily. Software has various quality-related characteristics. Moreover, there are various international standards for quality [2]. In reality, however, quality management is often limited to the level of resolving defects after they occurred. Therefore, the defect density of a delivered product, that is, the number of defects or the size of defect column are used in defining the quality of a software, which has become an industry standard today [4][5]. Therefore, a defect, or a software defect can be defined as the cause that makes a software work differently from its intended function according to the customers' needs and requirements.

This paper identifies defects to produce a reliable software and analyzes the relationship among different defects. Another goal of this paper is to design a defect trigger based on the findings. So when we archive resemblant project, we can forecast defect and prepare to solve defect by using defect trigger.

## 2 Related Work

### 2.1 Defect

A defect is an important diagnostic type representing the process and product. Since a defect is closely related to the quality of software, defect data is more important than a man-month estimation in several ways [10][11].

Furthermore, defect data is indispensable for project management. Large-scale projects may include several thousands of defects. These defects may be found in each stage of the project by many people. During the process, it often happens that the person that identifies and reports a defect is not the same person who corrects the defects [12].

Usually, in managing a project, efforts will be made to remove most or all of the defects identified before the final delivery of the software. However, this will make it impossible to report and resolve a defect in a systematic manner.

The use of an unofficial mechanism in removing defects may make people forget about the defect that was discovered. As a result, the defect may remain unresolved, or additional man-month may be required later. Therefore it is a basic requirement to record and trace the defects until they are solved. For this procedure, we will need information such as the symptom of the defect, position of the suspected defect, the discoverer, the remover and so on. Since a defect can be found in the work product of a project, it may have a negative effect on achieving the goal of project.

The information on defects includes the number of defects that are identified through various defect detection activities. Therefore, all defects that are discovered in the requirement examination, design examination, code examination, unit test, and other stage are recorded. The distribution data of the number of defects found in different stages of the project are used in creating the Process Capability Baseline [15]. Several explanations are also recorded in the PDB(Process Data Base) input item, which includes explanation on estimation and risk management.

## 2.2 Impact of Defect

By recording the identified defects, it is possible to focus on analyzing the number of defects, the defect ratio in the initial state and other issues of the project. Such defect traceability is one of the best implementation guidelines for project management [7][8].

**Table 1.** Defect Data.

Data	Explanation of data	Essential / selective
Project code	Project code in which defect is identified	Essential
Explanation	Explanation on defect	Essential
Module code	module code	selective
Program name	Name of program in which defect is identified	selective
Detected stage	Stage in which defect is detected	Essential
Driven stage	Stage in which defect is driven / origin	Essential
Type	Classification of defect	Essential
Seriousness	Seriousness of defect	Essential
Examination type	Examination type	selective
State	Present state of defect	Essential
Presenter	Name of discoverer of defect	Essential
Owner	Name of person owning defect	Essential
Submission date	Date when defect is submitted to owner	Essential
Closing date	Date when submitted defect is closed	Essential

The analysis is made based on the types required for making use of the defect data. In this case, additional measures must be taken besides merely recording defects. For example, in order to understand the software process level, it is necessary to distinguish defects detected during test activity and those discovered after shipping.

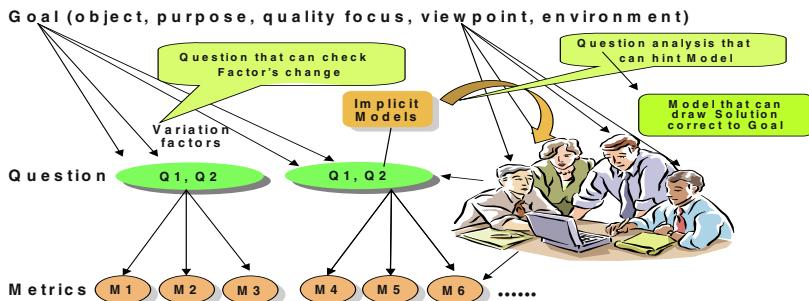
This kind of analysis becomes possible if the stages in which the defects were discovered are differentiated in order to make the base line to record the location and percentage of the defects and to compare predicted defects and actually identified defects [16].

One of the goals of such organizing is to improve quality and productivity through the continuous enhancement in process. Such approaches to improve quality and productivity include studies on the efficiency of defect elimination in various stages and on the possibility of additional improvement.

The DRE(Defect Removal Efficiency) of the defect detection stage is the ratio of the number of defects discovered at a stage against the total number of defects that appears when the stage is being processed. The more effective the DRE, the lower the possibility of undetected defects [4]. This shows that increasing the DRE(Defect Removal Efficiency) is a method to improve productivity. It is necessary to identify the location of the detected defect and the time the defect is inserted. In other words, it is crucial to understand the information of the stage in which the defect was inserted regarding each recorded defect.

### 2.3 GQM(Goal - Question - Metrics) Method

The GQM approach is defined in three stages, of which the composition is presented in Fig. 1.



**Fig. 1.** GQM method.

The first stage is a conceptual stage, of which the elements are Object, Purpose, Viewpoint, and Focus. This stage defines the Goal [23][24].

The second stage is an operational stage, in which the questions ask how much of the task is completed to achieve the preset goal [24].

In the third stage, quantitative answers are given to the question.

Through the above three stages, the system of the Metrics is completed.

### 2.4 Defect Tree Analysis

The Defect tree analysis builds a graphical model of the sequential and concurrent combinations of events that can lead to a hazardous event or system state. Using a well-developed defect tree, it is possible to observe the consequences of a sequence of interrelated failures that occur in different system components. Real-time logic

(RTL) builds a system model by specifying events and corresponding actions. The event-action model can be analyzed using logic operations to test safety assertions about system components and their timing. Petri net models can be used to determine the defects that are most hazardous[20].

If a risk is identified and analyzed, the requirement for safety can be specified on the software. In other words, the specification can include a list of undesired events and anticipated system responses. This is related with the software's role of managing undesired events.

Although software reliability and its safety are closely inter-related, it is important to understand the subtle difference between them. For determining software reliability, statistical analysis is used to calculate the probability of a software defect occurrence. However, the occurrence of a defect does not necessarily lead to a danger or disaster[21].

Software safety may be measured by the extent that a failure actually results in a disaster.

In other words, a failure is not evaluated in a vacant or isolated state but within the entire computer-base system.

## 2.5 Process Concepts

Process is what people do, using procedures, methods, tools, and equipment, to transform raw material (input) into a product (output) that is of value to customers. A software organization, for example, uses its resources (people, and material) to add value to its inputs (customer needs) in order to produce outputs (software products).

- Process.

A sequence of steps performed for a given purpose [25].

- Software Process.

A set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products [26]. Processes exist at various levels, and serve general or specific goals. At the organization level, processes interact broadly with the environment or seek organization-wide goals; at the tactical and operational levels, processes serve specific project or functional goals; at the individual level, processes accomplish specific tasks. The process management premise is that the quality of the product (e.g. a software system) is largely governed by the quality of the process used to develop and maintain it.

- Process context.

Organizations as systems with strategic, technical, structural, cultural, and managerial components; relation of process to other components of organizational systems; people, process, and technology as three quality leverage points; relating process and product; relating to external forces; process levels; formal and informal processes.

## 2.6 Process Definition

Process definition consists of adding and organizing information to a process model to ensure it can be enacted. A process is defined when it has documentation detailing what is done, who does it, the materials needed to do it, and what is produced. A software process definition establishes a plan for applying tools, methods, and people to the task of software development[27].

- Process definition activities.

Product planning, process familiarization, customer identification, interviewing, analysis, model construction, verification and validation.

- Components of software definition.

A software definition document will consist of information about work product, activity, and agent viewpoints. That is, the document identifies work products to be produced, activities, and the agents involved in producing the work products.

- Related terms and concepts.

Process design, process management principles, life-cycle-models, descriptive modeling, prescriptive modeling, organizational process asset, perspective viewpoint, process asset, process model, process guide.

## 2.7 Software Processes

Software processes have been categorized and structured in different ways. Two major process breakdowns are described below.

### 2.7.1 Process and It's Categories in the SPICE Baseline Practices Guide

The SPICE Baseline Practices Guide [28] documents the set of practices considered essential to good software engineering. The base practices are grouped into processes. Sets of processes that should be implemented to establish and improve an organization's software development, maintenance, operation, and support capabilities are organized into process categories that address the same general area of activity. Table 3 shows the five process categories and their member processes.

### 2.7.2 Key Process Areas in the Capability Maturity Model for Software (CMM)

“Each key process area identifies a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing process capability.” — CMM for Software

The CMM [26] presents a set of recommended practices in eighteen key process areas (KPAs) that have been shown to enhance software development capability [29]. Each KPA resides at a single maturity level. The 18 Key Process Areas have been categorized into three broad categories: management, organizational, and engineering

processes [26]. The maturity levels, KPAs, and categorizations are shown in Table 4. Note that at Levels 4 and 5 there are KPAs that span process categories, and that no KPAs are associated with the Initial Level.

### 3 Theory and Case Study

This chapter will develop a questionnaire to identify defects that occur during actual projects of companies.

Therefore, the structure and contents of the questionnaire to detect the defect and analyze the result are presented in this chapter. Furthermore a defect trigger based on the analyzed result is provided.

As for the domain of the project, digital systems were selected and 21 projects of four companies were chosen as target companies.

#### 3.1 Collecting Defect Data

The operational activities of measurement begin with collecting data. The procedures that are defined for collecting and retaining data need to be integrated into the software processes and be made operational.

Collecting data is more than mere measurements. It consists of implementing plans, sustaining the research work, and the resulting measurement activities. Documenting procedures involves:

- Identifying the responsible persons and organizations
- Specifying where, when, and how measurements will be conducted
- Defining procedures to be used for recording and reporting results

The data was collected by performing assessments with SPICE and by rating SEI maturity questionnaire for CMM/KPA.

The requirements help to ensure that the assessment output is self-consistent and that it provides evidence to substantiate the ratings.

The assessment shall be conducted following a documented process that meets the assessment purpose. The assessment process shall contain at minimum activities such as planning, data collection, data validation, process rating, and reporting.

The training and assessment of the ISO Guide 62 and TR 15504, part 6 supported the reliability of the assessment. Furthermore, the people who rated the maturity questionnaire have been screened in order to confirm the reliability of the assessment.

The certified assessors who conducted assessments of the concerned company with SPICE (ISO TR 15504, ver.3.3, 1998) also rated the SEI maturity questionnaires.

The project portfolio assessed is in the following table.

**Table 2.** Project portfolio.

Samples	Development members	KSLOC	Period of Development (Months)	Number of Assessors
A	191	150	7	10
B	302	100	6	10
C	1.763	130	10	12
D	1.325	80	8	11

### 3.2 Questionnaire to Detect Defects

A defect detection questionnaire was made to identify defects in the ongoing project of a company.

Another purpose is to discover defects and analyze the causes between each defect. Questions that must be examined in the questionnaire were divided as below by classifying defects that exist in each life cycle into a detailed category. The structure of the questionnaire for each life cycle is as follows.

To detect defects, five detailed categories were made for the questionnaire at the requirement analysis stage. Detailed categories for defect detection at the requirement analysis stage reflect customers' requirements and confirm how the requirements are satisfied. The requirements are classified in order to confirm consistency. Therefore, the detailed category consists of completeness, correctness, quality attribute, and traceability. For other items, an investigation was made in order to confirm whether the specification was based on the planned form and schedule that support the four-detailed category. Each detailed category is shown in table 3.

**Table 3.** Questionnaire structure in requirements analysis stage.

Life cycle	Detailed category	Question purpose
Requirement analysis	Completeness	Questions asking whether the algorithm is defined to satisfy validity of requirement
	Correctness	Questions examining whether requirement is embodied by function
	Quality attribute	Questions about items related to quality attribute to confirm whether requirement is satisfied
	Traceability	Questions to confirm consistency of requirement
	Etc.	Questions for other items that do not belong in the detailed category

The structure of the Defect detection questionnaire in the design stage is shown in table 4. The purpose of developing a detailed category is to confirm that customer's requirement in the design stage is well reflected, which is a prerequisite for the requirement to be correctly reflected in the implementation stage. Therefore the detailed

categories were divided according to how clearly the module specification in the design stage was defined both internally and externally. The category is composed of structure, data, correctness, standard, traceability, logic, interface, clearness and robustness.

**Table 4.** Questionnaire structure in design stage.

Life cycle	Detailed Category	Question purpose
Design	Structure	Questions asking whether the structure in the design phase is clear, and whether integration and examination are easy
	Data	Questions asking whether data is initialized to precisely define relation between modules(components)
	Correctness	Questions asking whether designing is carried out to satisfy requirement
	Standard and Traceability	Questions asking whether a standard process is used and whether the requirement stage is traceable during design
	Logic	Questions asking whether there is no logical error
	Interface	Questions asking whether interface is precisely defined
	Clearness	Questions asking whether consistency exists in product presentation in design phase and is clearly expressed
	Robustness	Questions asking whether processing in an exceptional situation is considered.

The questionnaire structure in the coding stage is presented in table 5. Classification of a detailed category confirms how the requirement is implemented reliably as a function in the implementation stage after it has gone through design stage.

When an error was identified by analyzing the relations between code interior and exterior, the detailed category was classified based on standards confirming how this error was detected and resolved.

**Table 5.** Questionnaire structure in coding stage.

Life cycle	Detailed category	Question purpose
coding	Method	Question asking whether the function is completely realized in implementation
	Static Relation	Question asking whether error exists in the internal contents of code
	Dynamic Relation	Question asking whether error is detected within connection of outside device and the relevant file is managed

The questionnaire structure in the test stage is presented in table 6. The purpose of the detailed category of the questionnaire is to confirm that the test is implemented on

schedule, and under the planned environment and standard. Another purpose is to confirm whether the test result satisfies customers' requirements and whether correct responsibilities and roles are assigned based on a planned resource allocation.

**Table 6.** Questionnaire structure in test stage.

Life cycle	Details Category	Question purpose
Test	Test plan	Questions asking whether test environment and objectives are planned by schedule
	Correctness and Completeness	Questions asking whether guidelines and standard are clearly defined for a precise and perfect test implementation
	Standard and Traceability	Questions asking whether test is being implemented based on requirements
	Regression test	Questions asking whether change and version of code are differentiated and are modified
	Resources and Schedule	Questions asking whether responsibilities and roles are assigned based on a scheduled resource allocation

The actual questionnaire includes 17 requirements, 28 designs, 11 items for coding and 26 tests. The surveyed person is required to answer the questions asking whether the function works correctly with "Yes" or "No". Even when the response is "Yes", a problem may occur later even if the function or standard asked exist.

Therefore, such cases were classified as a defect, and the ratio of the defect of the relevant question against total defects for each question item was calculated.

### 3.3 Data Analysis of Questionnaire to Detect Defects

Among the question items in the defect detection questionnaire, there were items to which developers commonly replied that there was a defect. These items were classified separately. When these question items are classified according to life cycle and category, they are as follows.

The analysis of the questionnaire in which defects existed in the requirement analysis stage is as follows. Each company was found to conduct design and implementation stages by specifying extracted requirements.

However, it has been discovered that even if the requirements change and are functionalized, the task to check whether the intended functions are satisfied seemed insufficient.

According to the question results, most companies were found to reuse the extracted requirements without modification.

Therefore, methods to extract the customer's requirement and to confirm whether there are additional changes in the requirement and to confirm whether the requirements are met are required.

Defects detected in the requirement stage are more important than defects detected in other development stages.

The defect ratio was 35% against the total, of which 25% was for completeness, and 10% for correctness. Although defects were also identified in the design, implementation and examination stages, it has been found out that that defects in the requirement stage were responsible for the defects in the following stages. Therefore, predicting defects and finding a solution in the initial stage of a project would have a positive effect in terms of schedule, cost, and quality of the project.

**Table 7.** Defects detected in requirements analysis stage.

category	Question contents
completeness	1. Is the written requirement described correctly in detail?
	2. Was the basic algorithm defined to satisfy functional requirements?
	3. Does software requirement specification include all the requirements of the customer and system?
	4. Was mutual confirmation of conformability with other requirements made?
correctness	1. Is a consistent delivery of a specified error message possible?

The contents of the defect detection questionnaire in the design stage are as follows. The design stage is a stage to systemize the extracted requirements so that they are functionalized in the system.

From analyzing the questionnaire answers, it is possible to say that a defect existed in the base structure for using the data in the design stage. Furthermore, in order to predict the mutual relationship among modules, it is necessary to clearly define the interface. The defect ratio in the design stage was 25% against the total defects according to the analysis. Among this 25%, 10% was for correctness, 5% for data, 5% for logic, and 5% for interface. The questionnaire for defects detected in the design stage is shown in table 7.

**Table 8.** Defects detected in design stage.

category	Question contents
data	1. Did you explain the content that was omitted in description of system data?
	2. Was all the data defined properly and was initialized?
correctness	1. Are reliability and performance requirements specified?
	2. Did undefined or unnecessary data structure exist?
	3. Did you consider all constraint items?
	4. Can you analyze to decide required productivity, response time, and correctness? Can you implement this in design?
logic	5. Can you verify in design stage?
interface	1. Are all interfaces defined clearly?

Defects in the coding stage were mainly functional errors that happen in a code. However, as the products near delivery to customers as each stage are processed after the coding stage, the companies analyzed the stages after coding in detail and prepared alternatives. However, defects that resulted from the defects in the requirement analysis and the design stage took up a large portion among defects in the coding stage.

The defect ratio in the coding stage was 15% out of total defects, among which 5% was for method, 5% for static relation, and 5% for dynamic relation.

The defect detection questionnaire in the coding stage is presented in table 9.

**Table 9.** Defects items detected in coding stage.

category	Question contents
Method	1. Is function associated with outside spec processed correctly ?
	2. Is the code re-useable by calling external reusability component or library function?
	3. Can you make one procedure by summarizing recursive code block?
Static relation	1. Is code documented in comment form for easy management?
Dynamic relation	1. Is the existence of a file confirmed before accessing the file?

In the test stage, defects were not applied with the tolerance limit and output standard of the defect during testing.

The resource and human resource allocation and role assignment in planning the test were not satisfactory.

According to the analysis, the defect ratio in the test stage was 25% out of total defects, among which 10% was for completeness and correctness, 5% for standard and traceability, 5% for revolution examination, and 5% for resources and schedule. The defect detection questionnaire in the test stage is shown in table 10.

The relations among defects of each stage that exist in the development process is presented in 3.4

### 3.4 Cause-Effect Analysis for Defect

Defects in respective stages are presented in 3.3.

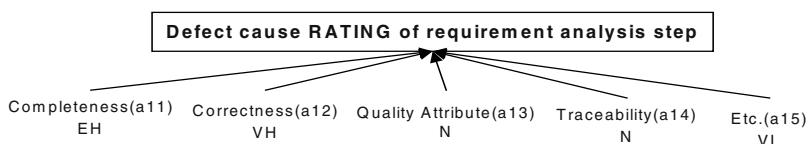
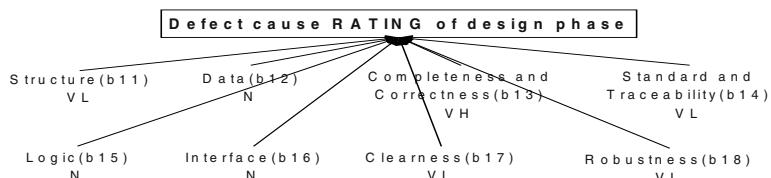
Therefore this chapter will analyze and measure the identified defect causes. To this end, the hierarchical structure for decision-making has been schematized, and the weight of the defect item in each stage has been graded into six dimensions. The graded items were analyzed based on a comparison metrics [21], and the geometric means of the defect items were calculated in order to identify its relation with the causes.

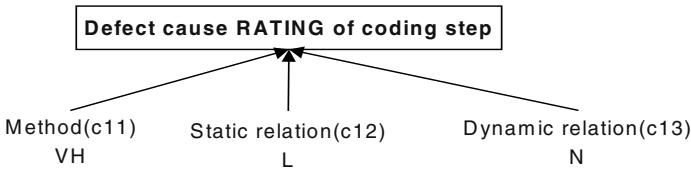
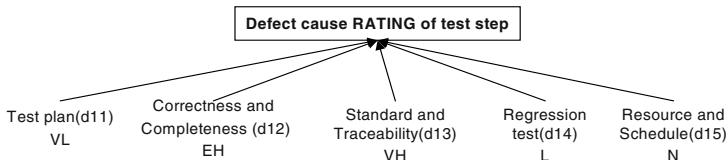
**Table 10.** Defect items detected in test stage.

category	Question contents
correctness and completeness	1. Were the target level of the requirement and code coverage specified quantitatively?
	2. Were the defect tolerance limit and output standard of the pass/fail in test plan determined?
	3. Was the integration test procedure explained in the interface of design document?
	4. Is the application range of test sufficient for reliability?
	5. Do the explanations of functions correspond exactly and correctly to the document of test plan?
	6. Are the beginning and end condition of the test practical ?
	7. Are there items that do not exist in test document?
	8. Was the delivery of test result defined?
	9. Is test plan perfect, precise, and concrete?
standard and traceability	1. Did the test plan contain all specifications, standard and development documents?
regression test	1. Are the tests distinguishable from previous tests and able to be utilized in order to use sufficient test examples?
	2. Is the code tested sufficiently when it is modified? Is partial modification of the interface possible?
	3. Did all resources consider human, hardware and software?
resource and schedule	2. Were development, acquisition of method and tool scheduled well?
	3. Were the related person's role and responsibility distinguished in test activity?
	1. Were the related person's role and responsibility distinguished in test activity?

### 3.4.1 Hierarchical Decision-Making Structure

The decision-making structure and graded items in each stage are as below.

**Fig. 2.** Cause rating scale of defect in requirement analysis stage.**Fig. 3.** Cause rating scale of defect in design stage.

**Fig. 4.** Defect cause rating scale in coding stage.**Fig. 5.** Defect cause rating scale in test stage.

The defect cause rating scale structure is divided into four stages (requirement analysis, design, coding, test), and each stage has been further categorized in detail.

The rating scale is divided into six dimensions: Very Low(VL), Low(L), Normal(N), High(H), Very High (VH), Extra High (EH). This means that, among the six dimensions, priority increases as it nears EH, and the priority decreases when as it approaches VL.

In the case of Fig. 2, at the requirement analysis stage, a rating scale was made based on the weight of each item analyzed in 3.3. The result showed that the completeness item had the most defects.

Therefore, completeness and correctness were found out to be greater causes of defect than quality attribute and traceability.

It has been identified that the defect cause in the design stage of Fig. 3 was correctness and completeness. The defect cause in coding stage of Fig. 4 was method, and the defect cause in test stage of Fig. 5 was correctness and completeness, standard and traceability.

### 3.4.2 Comparison Matrix for Defect Analysis

A matrix was used to compare the defect causes. The matrix shows the relation by listing each importance from 1 to 9 [20].

The measurement standard table of the Comparison Matrix is shown in table11.

The importance among the items of each stage were quantified based on table 11.

Fig. 2, 3, 4, 5 were analyzed by table 11. This time, the differences of the item grade of each stage were compared. For example, if an item is Extra High(EH) and the other item is N, the importance between the two items is triple the difference. This is due to two grade differences. The same relation applies to table 12.

**Table 11.** Comparisons Matrix measurement standard table.

(a <sub>ij</sub> )	Definition	
1	Less importance	Equal importance
3		
5		
7		
9		Extreme importance
2, 4, 6, 8	Intermediate value	
Reciprocal	If $a_{ij}=w_i/w_j$ , then $a_{ji}=1/a_{ij} = w_j/w_i$	

**Table 12.** A ratio production table of Grade vs importance difference.

Grade difference between item	Grade vs importance difference ratio
1	Double
2	Triple
3	Quadruple
4	Quintuple
5	Sextuple

**Table 13.** Comparative matrix between items of requirements analysis stage.

	a11	a12	a13	a14	a15
a11	1	2	4	4	6
a12	1/2	1	2	2	4
a13	1/4	1/2	1	1	2
a14	1/4	1/2	1	1	2

By table 12, it is possible to produce a comparison matrix of Fig. 1. The contents are the same with table 13.

In table 13, the importance was compared with other items based on a11, a12, a13, a14, a15 columns.

For example, according to the analysis, between a11 and a12, a11's defect item is double that of a12's. This is because a11 is EH, and a12 is VH. Therefore, the difference of grade between items is 1 by table 11. Therefore, the difference between the grade and importance is double. This implies that the probability that a11 will cause a defect is twice as high as a12.

The comparison matrix of the design, coding, test stage were produced by the same method, of which the contents appear in table 14, 15, 16.

**Table 14.** Comparative matrix between items of design stage.

	b11	b12	b13	b14	b15	b16	b17	b18
b11	1	1/3	1/5	1	1/3	1/3	1	1
b12	3	1	1/3	3	1	1	3	3
b13	5	3	1	5	3	3	5	5
b14	1	1/3	1/5	1	1/3	1/3	1	1
b15	1/3	1	1/3	3	1	1	3	3
b16	1/3	1	1/3	3	1	1	3	3
b17	1	1/3	1/5	1	1/3	1/3	1	1
b18	1	1/3	1/5	1	1/3	1/3	1	1

**Table 15.** Comparative matrix between items of coding stage.

	c11	c12	c13
c11	1	4	3
c12	1/4	1	½
c13	1/3	1	1

**Table 16.** Comparative matrix between items of test stage.

	d11	d12	d13	D14	d15
d11	1	1/6	1/5	1/2	1/3
d12	6	1	2	5	3
d13	5	1/2	1	4	3
d14	2	1/5	1/4	1	½
d15	3	1/3	1/3	2	1

### 3.4.3 Calculation of Defect Cause Priority

One of the purposes here is to calculate the importance of each stage item based on the comparison metrics from 3.4.2. The greater the importance of each item, the higher the probability of causing a defect. This was calculated by geometric means. A production table of each stage is as follows.

**Table 17.** Defect cause importance table of requirement analysis stage.

Item	Production value
a11	(1x2x4x4x6)1/5 = 1921/5 = 2.862
a12	(1/2x1x2x2x4)1/5 = 321/5 = 1.6
a13	(1/4 x1/2x1x1x2)1/5 = 0.251/5 = 0.758
a14	(1/4x1/2x1x1x2)1/5 = 0.251/5 = 0.758
a15	(1/6x1/4x1/2x1/2x1)1/5 = 0.101/5 = 0.631

**Table 18.** Defect cause importance table of design phases.

Item	Production value
b11	$(1 \times 1 / 3 \times 1 / 5 \times 1 \times 1 / 3 \times 1 / 3 \times 1 \times 1) / 8 = 0.0071 / 8 = 0.538$
b12	$(3 \times 1 \times 1 / 3 \times 3 \times 1 \times 1 \times 3 \times 3) / 8 = 271 / 8 = 1.510$
b13	$(5 \times 3 \times 1 \times 5 \times 3 \times 3 \times 5 \times 5) / 8 = 168751 / 8 = 3.376$
b14	$(1 \times 1 / 3 \times 1 / 5 \times 1 \times 1 / 3 \times 1 / 3 \times 1 \times 1) / 8 = 0.0071 / 8 = 0.538$
b15	$(1 / 3 \times 1 \times 1 / 3 \times 3 \times 1 \times 1 \times 3 \times 3) / 8 = 31 / 8 = 1.147$
b16	$(1 / 3 \times 1 \times 1 / 3 \times 3 \times 1 \times 1 \times 3 \times 3) / 8 = 31 / 8 = 1.147$
b17	$(1 \times 1 / 3 \times 1 / 5 \times 1 \times 1 / 3 \times 1 / 3 \times 1 \times 1) / 8 = 0.0071 / 8 = 0.538$
b18	$(1 \times 1 / 3 \times 1 / 5 \times 1 \times 1 / 3 \times 1 / 3 \times 1 \times 1) / 8 = 0.0071 / 8 = 0.538$

**Table 19.** Defect cause importance table of coding stage.

Item	Production value
c11	$(1 \times 4 \times 3) / 3 = 121 / 3 = 2.288$
c12	$(1 / 4 \times 1 \times 1 / 2) / 3 = 0.1251 / 3 = 0.5$
c13	$(1 / 3 \times 2 \times 1) / 3 = 0.6671 / 3 = 0.874$

**Table 20.** Defect cause importance table of test stage.

Item	Production value
d11	$(1 \times 1 / 6 \times 1 / 5 \times 1 / 2 \times 1 / 3) / 5 = 0.0061 / 5 = 0.306$
d12	$(6 \times 1 \times 2 \times 5 \times 3) / 5 = 1801 / 5 = 2.825$
d13	$(5 \times 1 / 2 \times 1 \times 4 \times 3) / 5 = 301 / 5 = 1.974$
d14	$(2 \times 1 / 5 \times 1 / 4 \times 1 \times 1 / 2) / 5 = 0.051 / 5 = 0.549$
d15	$(3 \times 1 / 3 \times 1 / 3 \times 2 \times 1) / 5 = 0.6671 / 5 = 0.922$

Items of each stage were produced by a geometric average. The result of the analysis showed that, in the requirement analysis stage, all(completeness) and a12(correctness) were the defect causes.

As for the design stage, the analysis is that b12(data), b13(correctness and completeness) and b15(logic), b16(interface) items can become defect causes.

In the coding stage, it has been found out that c11(method) is the greatest defect cause, and c13(dynamic relation) can also become a defect cause.

In the test stage, d12(correctness and completeness) and d13(standard and traceability) are the greatest defect causes, and d15(resources and schedule) is a possible defect cause.

We schematized the defect trigger in 3.5 using the above analyzed data.

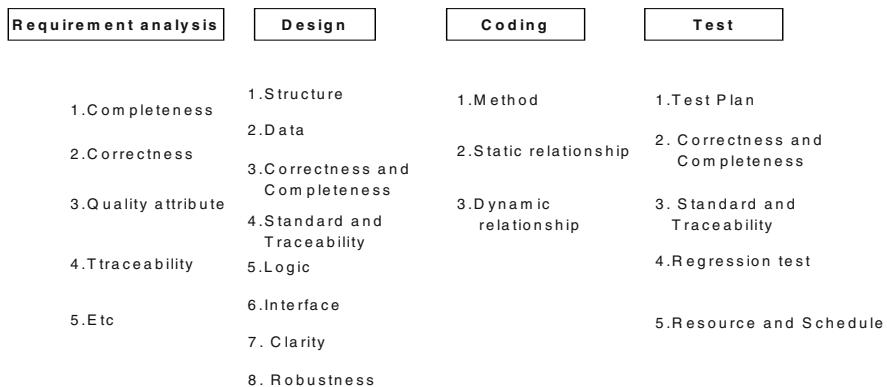
### 3.5 Defect Trigger

There are many tools today that are used to define the relations among cost, schedule, and quality. Among them, those that focus on the relations among defects are quality models COCOMOII and COQUALMO, which are under extension and further development at present. Among the two, especially COQUALMO focuses on detecting and removing defects [3].

A defect detection questionnaire was developed to ask questions of companies in order to analyze defects. Therefore, the defects of each development stage were detected, and the result was analyzed. Also, in order to prevent a defect, a defect trigger based on the analysis of detected defects is presented.

The defect trigger distinguishes the different causes of defects and finds the relation between defects, thereby revealing the cause of a defect. Therefore, defect data must be used to complete a defect trigger. By using a defect trigger in a similar project, it is possible to identify the relation between defects and predict possible defects, which ultimately will be a useful tool for defect prevention.

In the previous chapter 3.4, in order to design the defect trigger, the triggers items of each development stage were presented according to a detailed category explained in chapter 3.1. The structure of the defect trigger is as follows.



**Fig. 6.** Structure of defect trigger.

By using a defect trigger, it is possible to analyze the causes of defects in all previous stages (requirement analysis, design, coding and test stage).

According to the result of the analysis, identified the relation between defects and software defect process.

The trigger about the relation between defects is as follows.

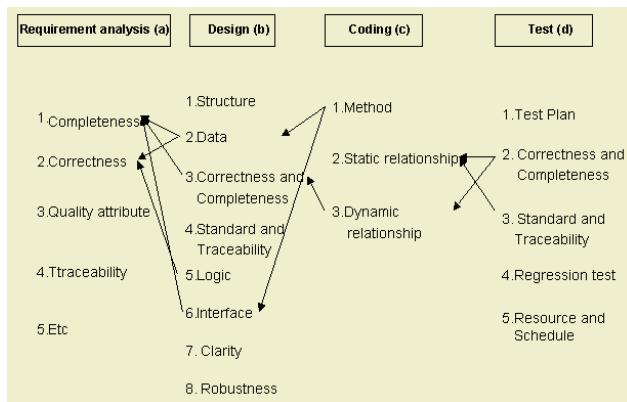


Fig. 7. Association trigger between defects.

The defect of data category in the design stage is attributable to the fact that the definition of the data content were not analyzed in detail in the requirement analysis stage. In addition, as for defects involving correctness and logic category, the error could not be passed on correctly, since the data contents were lacking. The causes of defects in the correctness category are related to completeness in the requirement analysis stage, and those in the logic category were found to be related to the correctness category.

Because the conformability between requirements was not confirmed sufficiently, the interface category could not be clearly defined. Therefore, it has been analyzed that the completeness category of in the requirement stage to be the causes of these defects.

The defects in the coding stage were caused by data in the design stage and interfacing problems. The reason is that there is no detailed explanation about the repetition codes and the relationship with external specification, and that the distinction of inter-relationships through interfacing is lacking. The causes of defect in the dynamic relation category were found to be the inaccuracy in the design stage due to insufficient consideration of restriction.

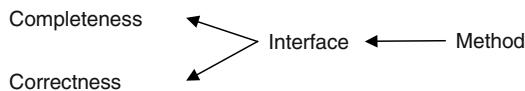
The defect causes of correctness and completeness category in the test stage were due to insufficient consideration of documentation and constraint.

Therefore, the defect causes of the correctness and completeness category at the coding stage were attributable to faults in the static and dynamic relation category. The defect causes of the standard and traceability category resulted from faults in the static relation category at the test stage, due to lacking documentation.

In conclusion, by removing defects in the completeness and correctness category at the requirement stage, the defects of data, correctness, logic and interface can be prevented at the design stage. In this regard, when expressing a causal relationship in the trigger in a picture, it is as follows.



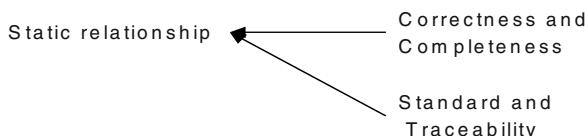
**Fig. 8.** Defect trigger-1 of details category.



**Fig. 9.** Defect trigger-2 of details category.



**Fig. 10.** Defect trigger-3 of detailed category.



**Fig. 11.** Defect trigger-4 of detailed category.

## 4 Introduction to Effectiveness Analysis of Defect Trigger

We analyzed aspect that defect removal is achieved efficiently to analyze effectiveness of defect trigger.

Also, we present effectiveness of introduction through if productivity improves because defect of whole project is reduced through defect trigger.

#### 4.1 Defect Removal Efficiency Analysis

Purpose of Defect Trigger design improves quality of product and heighten productivity.

Therefore, when we applied Defect Trigger in actuality project, we wish to apply defect exclusion efficiency (Defect Removal Efficiency). to measure ability of defect control activity.

After apply Defect Trigger, defect exclusion efficiency analysis [20] investigated defect number found at relevant S/W development step and defect number found at next time step in terms of request analysis, design and coding stage. Production of defect exclusion efficiency is as following.  $DRE = E/(E+D)$

$E$ = Number of defect found at relevant S/W development step(e.g : Number of defect found at request analysis step)

$D$ = Number of defect found at next S/W development step (e.g : Defect number that defect found at design step is responsible for defect of request analysis step)

Ideal value of DRE is 1, and this displays that any defect does not happen to S/W.

**Table 21.** Table of defect removal efficiency.

	Number(%) of defect found at relevant S/W development step (E)	Number(%) of defect found at next S/W development step (D)
Requirement	10	3
Design	15	3
Coding	5	1

Table 21 is a table to inspect S/W development step defect number after Defect Trigger application.

$$0.769 = 10/(10+3) \text{ (Requirement phase)}$$

$$0.833 = 15/(15+3) \text{ (Design phase)}$$

$$0.833 = 5/(5+1) \text{ (Coding phase)}$$

If we save DRE at each S/W development step by Table 21, it is as following. Therefore, because DRE is approximated to 1, when we remove defect by Defect Trigger, defect exclusion efficiency was analyzed high.

#### 4.2 Size-Based Software Metrics

After apply Defect Trigger, we investigate by Table 22 to compare and analyze how productivity improved with last project[20].

**Table 22.** Size based software metrics.

	Last project	The present project
SLOC	40,000	120,620
Project Cost	400,282,000	1,500,000,000
Effort (Man-Month)	55.2	381.6
Defect number	400	810
Project People	11	50

If depend to Table 22, last project decreased more remarkably than number of defect found in the present project.

And in case defect happens, it is decreased effort (Man-Month) and human strength to solve this.

Being proportional in scale of project, Contents of each item are increasing. Therefore, based on whole SLOC, project expense and Effort(Man-Month), we compared number of found defect. By the result, scale of project increased by 30% than previous project but number of found defect decreased by 20% than whole scale.

Therefore, this brought productivity elevation by Defect Trigger.

## 5 Conclusions

On the Fig.7 we can represent 4 stages as a,b,c,d, and the trigger elements of each stage as  $a_i$ ,  $b_i$ ,  $c_i$ ,  $d_i$ ,  $i = 1, n$  on requirement phase.

Partially ordered set are as follows;

$a_1 < b_2$	$a_2 < b_2$	$b_2 < c_1$	$b_3 < c_3$	$b_6 < c_1$	$c_1 < d_2$	$c_3 < d_2$
$a_1 < b_3$	$a_2 < b_5$				$c_2 < d_3$	
$a_1 < b_6$						

The term “poset” is short for “partially ordered set”, that is, a set whose elements are ordered but not all pairs of trigger elements are required to be comparable in the order as:

$$a_i < b_j, b_i < c_{j, ci} < d_j \text{ for some } i, j$$

Just as an order in the usual same may be strict ( as  $<$  )

A strict partial order is a relation  $S$  on a set  $X$  satisfying the following conditions:

(R) (Irreflexivity) for no  $x \in X$  does  $(x, x) \in S$  hold;

(A) (Antisymmetry) if  $(x, y) \in S$ , then  $(y, x) \notin S$ ;

(T) (Transitivity) if  $(x, y) \in S$  and  $(y, z) \in S$ , then  $(x, z) \in S$

How many different posets are there with  $n$  elements?

A total order is a partial order in which every pair of trigger elements is comparable, that is, the following condition holds;

For all  $x, y \in X$ , exactly one of  $x <_R y$ ,  $x=y$ , and  $y <_R x$  holds.

In a poset  $(X, R)$ , we define the interval  $[x, y]_R$  to be the set,

$$[x, y]_R = \{ z \in X : x <_R z <_R y \}$$

And so we can make total order, so called topological sort of all of the trigger elements over the all stager and using the total order we can study cause analysis about defect.

The problem of topological sorting is to “embed the partial order in a linear order,” to arrange the elements into a linear sequence  $a_i, b_j, c_k, \dots, z_l$  such that whenever  $a_j < b_k$ , we have  $j < k$ [30].

Chapter Three has presented the designing and analysis of a defect trigger based on defect causes.

In implementing a similar project, using such triggers will help produce a more reliable result in terms of cost, quality, and schedule of the entire project by predicting the stages where defects occur and the contents of the defects.

In order to make this possible, further study is required on how this can be used to enhance the COQUALMO defect reduction estimates by using the trigger information.

The direction of further study should be to develop a defect trigger framework by providing detailed items of the defects, and to develop a system that is capable of predicting defects by inputting major items on the web.

This work has been written based on the USC/CSE International Forum.

## Acknowledgement

We would like to express our sincere gratitude to all the participants of the affiliate program. We would like to express special thanks to Dr. Boehm.

## References

1. Kyung-Whan Lee, "Modelling for HDC", The 5th Korean Conference on Software Engineering, Korea information science society, Feb. 20, 2003
2. Annual Research Review, USC/CSE workshop reports, oct. 2002
3. Barry W. Boehm, "SOFTWARE COST ESTIMAITION WITH COCOMO II", Prentice-Hall PTR, 2000
4. Software Process Improvement Forum, KASPA SPI-7, dec. 2002
5. George Eckes, The Six Sigma Revolution, John Willey & Sons, 2001
6. SPICE Assessment in korea, The korea SPICE, May 13, 2002
7. Kyoung-Ae Shin, "Research about software reliability development model that defect importance is considered", journal of the korea computer industry education society, Vol. 03 No. 07 pp. 0837-0844, 2000. 7.
8. Robert H. Dunn, "Software defect removal", McGraw-hill, 1984.
9. Ram Chillarrege, Kothanda R. Prasad, "Test and Development Process Retrospective ? a Case study using ODC Triggers", IEEE computer Society, 2002. 4
10. N. Fenton and N. Ohlsson "Quantitative analysis of faults and failures in a complex software system", IEEE Trans. Software Eng., 26, 797-814, 2000.
11. McCall, P.K. Richards and G.F. Walters, "Factors in software quality." Vol 1, 2 and 3. Springfield VA., NTIS, AD/A-049-014/015/055, 1997
12. Vouk, Mladen, A., "Software Reliability Engineering", Tutorial Notes? Topics in Reliability & Maintainability & Statistics, 2000 Annual Reliability and Maintainability Symposium, Los Angeles, CA, 2000 January 24-27.

13. Padberg, "A Fast Algorithm to Component Maximum Likelihood Estimates for the Hypergeometric Software Reliability Model", Asia-Pacific Conference on Quality Software APAQS 2 40-49. 2001.
14. Vamder Wiel, Votta, "Assessing Software Designs Using Capture-Recapture Methods", IEEE Transaction on Software Engineering 1045-1054. 1993.
15. Wohlin, Runeson, "Defect Content Estimations from Review Data", Proceedings International Conference on Software Engineering ICSE 400-409, 1998.
16. Gaffney, John, "Some Models for Software Defect Analysis", Lockheed Martin, Nov 1996.
17. L.Hatton, "Is Modularization Always Good Idea", Information and Software Technology, Vol. 38, pp. 719-721. 1996.
18. B. Compton and C. withrow, "Prediction and Control of Ada Software Defects," J. Systems and Software, Vol. 12, pp. 199-207, 1990.
19. N. Fenton and M. Neil, "Software Metrics: Successes, Failures, and New Directions," J. Systems and Software, Vol 47, pp. 149-157, 1999.
20. Roger S. Pressman "Software Engineering" Mcgraw-Hill International edition, 1997.
21. Actin Focused Assessment for Software Process Improvement, Tim Kasse, Artech House, 2002
22. V. Basili, G. Caldiera and D. Rombach, "The Experience Factory", Encyclopedia of Software Engineering" Wiley 1994.
23. V. Basili, G. Caldiera and D. Rombach, "The Goal Question Metric Approach". Encyclopedia of Software Engineering. Wiley 1994.
24. Victor R. Basili, "The Experience Factory and its Relationship to Other Improvement Paradigms", Lecture Notes in Computer Science 717, Software Engineering ESEC'93, 4th European Software Engineering Conference Garmish-Partenkirchen, Germany, September 1993. December 1991
25. ANSI/IEEE Std 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology," February 1991.
26. Pault, Mark C.; Curtis, Bill; Chrissis, Mary Beth; & Weber, Charles V. Capability Maturity Model for Software, Version 1.1 (CMU/SEI-93-TR-24, ADA263403). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, February 1993.
27. Osterweil, Leon. "Software Processes are Software Too," 2-13. Proceedings of the 9th International Conference on Software Engineering. Monterey, CA, 30 March - 2 April 1987. Los Alamitos, CA: IEEE Computer Society, 1987.
28. SPICE Baseline Practices Guide (BPG) Version 1.00 (SPICE Project ISO/IEC ITC1/SC7/WG10). Internal draft, limited distribution. September 1994. (defines the goals and fundamental activities that are essential to software engineering, structured according to increasing levels of process capability)
29. Herbsleb, J.; Carleton, A.; Rozum, J.; Seigel, J.; & Zubrow, D. Benefits of CMM-Based Software Process (CMU/SEI-94-TR-13). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1994.
30. Thomas Britz, Peter Cameron, Partially ordered sets, J.of Formalized Mathematics, vol.1, 2002,Inst.of computer science, Univ. of Bialystok

# Certification of Software Packages Using Hierarchical Classification

Jaewon Oh<sup>1</sup>, Dongchul Park<sup>1</sup>, Byungjeong Lee<sup>2,\*</sup>, Jongwon Lee<sup>1</sup>,  
Euyseok Hong<sup>3</sup>, and Chisu Wu<sup>1</sup>

<sup>1</sup> School of Computer Science and Engineering, Seoul National University, Seoul, Korea  
`{jwoh,tongss,ljw,wuchisu}@selab.snu.ac.kr`  
Tel: +82-2-880-6573

<sup>2</sup> School of Computer Science, University of Seoul, Seoul, Korea  
`bjlee@venus.uos.ac.kr`  
Tel: +82-2-2210-5614  
Fax: +82-2-2210-5274

<sup>3</sup> School of Computer Science and Engineering, Sungshin Women's University, Seoul, Korea  
`hes@cs.sungshin.ac.kr`  
Tel: +82-2-920-7369

**Abstract.** Various types of COTS (Commercial-Off-The-Shelf) software are produced radically and its application areas are also being expanded. Because software quality evaluation depends on types of software and environments where software is operated, certification organizations need different certification programs that are foundations or basic frames for certifying software. We can certify new software products by making new certification programs or referring to previous ones typically. However, it is redundant to make new programs whenever certifying products and merely referring to previous ones may also cause problems because the previously certified software may not have the same quality characteristics as the new software has. For these reasons, we need to systematically derive certification programs for assessing new software, while considering types of software and software environments. Therefore, in this paper, we propose a meta model for systematically generating certification programs from previous ones. By using this model, certification programs are incrementally constructed based on hierarchical classification of software packages. This meta model is validated by generating certification programs with quality data on some certified software products.

## 1 Introduction

Software is playing an important, and often vital, role in almost all sectors of an economy increasingly. As a consequence, the production of high quality software has become a critical factor in the competitiveness of all sectors of industry. The ultimate goal of certification is to assess the quality of a software product and to create the warranty defining levels of quality that the user can expect of that product [1].

---

\* Corresponding author.

Evaluation of off-the-shelf software packages such as word processors, spreadsheets, and presentation software is useful for both suppliers and consumers. A vendor is interested in obtaining a certificate as a marketing tool, which may be essential for government procurement. A consumer of a software product seeks assurance about product quality. An independent certification body can provide such assurance by issuing a certificate.

Certification is a complex process. ISO/IEC (International Organization for Standardization / International Electrotechnical Commission) defines certification as a procedure by which a third party gives written assurance that a product, process, or service conforms to specified characteristics in ISO/IEC Guide 2.

Certification of software products by independent testing laboratories has been practiced since early 1990s, especially in Europe and recently in the United States [2]. There are two distinct approaches to software certification: process maturity assessment of an organization and product assessment [1, 3].

The first approach, process maturity assessment is based on the hypothesis that organizations with better organized teams and more advanced software engineering methodologies produce higher quality products. The certificate is granted by an independent certification body after it has audited an organization and found that there is an effective quality management system followed. ISO 9000 series, ISO/IEC 15504 (also known as Software Process Improvement and Capability dEtermination (SPICE)), CMM (Capability Maturity Model), and TickIT scheme are very commonly used in the software industry for process assessment and certification. The limitation of this approach is lack of proof of the final product quality.

The second approach and the focus of this paper is product certification. This approach assesses the software itself to determine its quality. Analysis is done toward a predefined set of characteristics by means of appropriate techniques to various product parts: specifications, source code, and user manuals [3]. ISO/IEC 9126 defines such characteristics that can be considered as the basis for certification of a software product [4-7]. For example, NSTL (National Software Testing Labs), GGS (Global Gaming Services), and DELTA (Danish Electronics, Light and Acoustics) are independent certification organizations that perform product certification based on ISO/IEC 9126.

We can perform software certification in any one of these two ways, but much work (e.g., [2-3, 8]) considers the assessment of product quality as well as the quality of the process used to create the product.

Typically, quality attributes and metrics may be relevant to a specific application domain; a study indicated that different types of software applications have different quality profiles [9]. And products can have quality in relation to their intended purpose [10]. Certification is a situation-dependent activity and requires the assessment of many quality attributes specifically created for a particular software product and for a particular set of intended uses [2]. Creation of a quality model including these attributes and metrics is not trivial and requires significant experience and knowledge about the subject matter being evaluated.

Various types of COTS software are produced radically and its application areas are also being expanded. Because, as mentioned above, software quality evaluation

depends on types of software and environments where software is operated, certification organizations need different certification programs or methodologies that are foundations or basic frames for certifying software and that include quality models for software. Also because of the rapid growth of the software industry, it is necessary to make certification programs promptly.

We can certify a software product by making new certification programs or referring to previous ones typically. However, it is redundant to make new programs whenever certifying a product and merely referring to previous ones may also cause problems because the previously certified software may not have the same characteristics as the new software has. For these reasons, we need to rapidly derive certification methodologies for assessing new software, while considering types of software and software environments. In this paper, we propose a meta model for systematically generating and maintaining such certification programs. By using this model, certification programs are incrementally constructed based on hierarchical classification of software packages.

This paper is organized as follows. In Section 2, software quality standards and software classification schemes are described. In Section 3, the meta model is proposed for systematic creation and maintenance of certification programs. In Section 4, the meta model is validated by generating some certification programs and the conclusions follow in Section 5.

## 2 Related Work

In this section, software quality standards that are used worldwide are described. Then, software classification schemes are described.

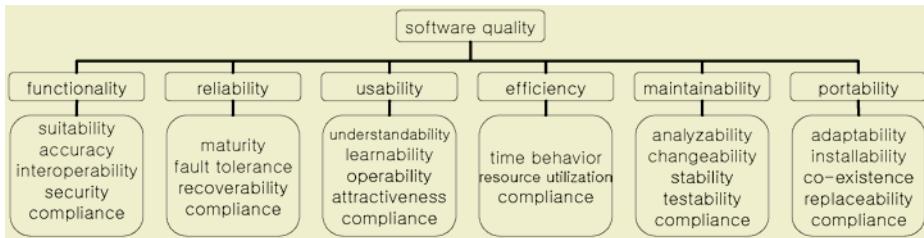
### 2.1 Software Quality Standard

In ISO/IEC 9126, software quality is defined to be the totality of characteristics of a software product that bear on its ability to satisfy stated or implied needs.

#### ISO/IEC 9126

In ISO and IEC, there are several standards for software quality specification and evaluation. Especially, ISO/IEC 9126 defines a quality model that is applicable to every kind of software. A quality model is a model that deploys quality into a set of characteristics and shows the relationships among them [11]. It provides the basis for specifying quality requirements and evaluating quality of a product.

ISO/IEC 9126-1 [4] proposes quality models in order to describe internal quality, external quality, and quality in use. Software quality is decided by referring to many different characteristics. In ISO/IEC 9126-1, a characteristic is defined as an element to which is referred to specify or evaluate software product quality. Each characteristic can be divided into subcharacteristics (Figure 1). In ISO/IEC 9126-1, there is not



**Fig. 1.** ISO/IEC 9126 quality model.

any specific method described to measure the characteristics or subcharacteristics. The detailed methods are proposed in ISO/IEC 9126-2 [5], ISO/IEC 9126-3 [6] and ISO/IEC 9126-4 [7].

In ISO/IEC 9126-2, external metrics are proposed as methods to quantitatively measure characteristics or subcharacteristics describing external quality. Similarly, ISO/IEC 9126-3 and ISO/IEC 9126-4 propose internal metrics and quality in use metrics to measure internal quality and quality in use, respectively.

The ISO/IEC 9126 standard is used primarily to describe software quality requirements. It can be also used to measure quality of software that is being developed or already published. Internal metrics are applicable to a non-executable software product during early stages of the development process [6]. These early measures are used by project managers as indicators of what to expect during testing and production. External metrics are applicable to an executable software product during testing or operating in later stages of development [5]. External metrics are used during testing as early predictors of the software quality that can be expected when the software is in operation. In this paper, certification of software packages by independent testing laboratories is considered. Thus, external metrics are used basically to evaluate software quality, since software publishers generally provide executables and manuals only.

### ISO/IEC 14598

ISO/IEC 14598 [12-17] gives an overview of software product evaluation processes and provides guidance and requirements for evaluation. The general process of quality evaluation consists of four procedural steps such as evaluation requirements establishment, the evaluation specification, design, and execution. The evaluation process provided by the ESPRIT Project SCOPE (Software Assessment and Certification Program Europe) [8] is the basis for that of ISO/IEC 14598.

When specifying the evaluation, using the internal or external metrics of ISO/IEC 9126 is recommended. ISO/IEC 14598-3 [14] proposes a developer's quality evaluation process. ISO/IEC 14598-4 [15] proposes a user's process, and ISO/IEC 14598-5 [16] proposes an evaluator's process which can be used by independent testing laboratories, when providing software product evaluation services.

To execute evaluation requires evaluation techniques. A first investigation and experiences with those techniques were made during the ESPRIT Project SCOPE [18].

The evaluation modules document what metrics and related techniques the evaluator has at their disposal [19]. This concept of an evaluation module is defined in 14598-6 [17].

### **ISO/IEC 12119**

ISO/IEC 12119 [20] states quality requirements for software packages: requirements for the product description, the user documentation, and programs and data. It also gives instructions how to test a software package for the compliance for the quality requirements. Testing is mainly done by inspection of documents and black box testing of programs and data. The standard may be used for issuing a certificate for software packages by third-party testing laboratories.

## **2.2 Software Classification**

Software classification should be done first since quality evaluation techniques should be chosen differently upon the kind of software. Unfortunately, there is no international standard for software classification. For this reason, each country or organization has its own classification scheme. ISO/IEC 12182 [21] describes the concept of a categorization of software although the document does not propose a specific software classification scheme.

Systematically grouping software into classes that specify the allowable uses for the software is particularly complex [22]. Various methods to classify software have been proposed. Of these methods, there are four major categories: enumerated classification, attribute-value classification, faceted classification, and free-text keyword indexing [23].

In enumerated classification, a subject area is broken into mutually exclusive, usually hierarchical, classes, much like the UNIX file system. An example of enumerated classification is the Dewey decimal system for library classification.

In attribute-value classification, parts are described in terms of a set of attributes and their values. For example, if an attribute is “version number,” its value might be “2.0.” The attribute can take any value assigned by the person classifying the part.

In faceted classification, parts are described in terms of a set of terms, or facets, and facet values. Faceted classification is similar to attribute-value method. However, with faceted classification the choice of values is limited. For example, if a facet is “operating system,” its value can only be one of Windows 98, Linux, and Unix.

Free-text indexing methods use the text from a document for indexing.

Each method has strengths and weaknesses. Enumerated classification has been applied by many organizations, is well understood by most people, and is easy to use. Therefore, our software classification scheme is based on enumerated classification. The difference is that classes may not be mutually exclusive in our scheme: one subclass can belong to one or more classes.

### 3 Meta Model for Software Certification

In this section, a meta model for software certification is proposed.

#### 3.1 Definition of Meta Model

A certification meta model is a model that can be referred to by certification organizations to describe certification models including certification policies that can be used to certify and measure software. Also, the meta model includes methods for systematically generating and maintaining certification programs. Before certifying software, certification organizations make certification models based on the meta model and then use them throughout the whole certification processes.

We need to certify software products considering their platforms, application areas, and environments. Thus, we identify the following three components of software package certification: software domain, quality factor, and software environment.

#### Software Domain

A software domain means a software classification system. Note that quality characteristics and metrics for evaluating and certifying software products vary according to properties of the products [1, 9]. Therefore, for certifying software products correctly, we need to classify products into classes and provide each class with relevant evaluation techniques including characteristics and metrics. To classify software systematically and efficiently, we identify the following requirements that a software domain should satisfy:

- Each and every software product should be classified into a group by a software classification system. Also, software products in one group should be able to be compared with each other.
- In Section 2, it was assumed that we use a hierarchical classification method for classifying software in a meta model: software products are classified into hierarchical groups. In a hierarchy of software groups, a lower group is a subset of its higher group. A lower group inherits properties of its higher group, such as necessary metrics and the quality level required, including new properties that are specific to the lower group. Note that the hierarchical classification structure is not a tree but a graph. So, one group can belong to one or more higher groups and as a result of that, multiple inheritance is allowed in the meta model.
- The more develops the world, the more produced new software is. Therefore, a classification system must not be fixed and should be flexible so as to classify new software effectively.

#### Quality Factor

A quality model provides the basis for specifying quality requirements and evaluating quality of a product. Software quality is not decided by one quality attribute but by many ones. A quality factor is a basic unit of quality evaluation and specification.

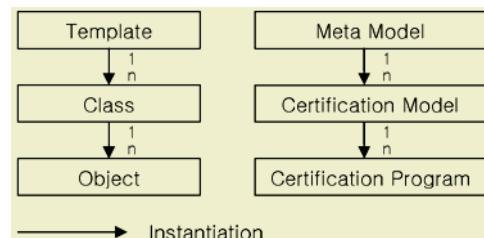
The characteristics of ISO/IEC 9126 can be examples of quality factors. As shown in Figure 1, in ISO/IEC 9126, software quality consists of six characteristics and each characteristic is divided into subcharacteristics. Software certification is performed by measuring quality factors.

## Software Environment

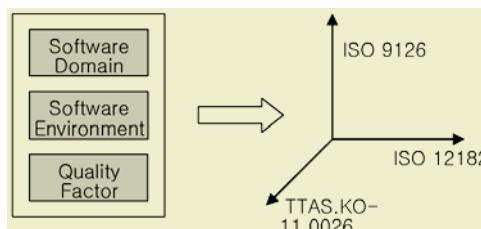
Because metrics for evaluating software quality are dependent on software environments, when measuring software quality, we should consider not only software itself but also real environments where the software is used [2, 10]. The software environment includes a platform, user class, operation mode, application area, etc. The software environment specifies a valid scope of quality certification. For example, a software product certified in a general embedded environment may not be suitable for a safety critical environment such as medical and aviation environments.

### 3.2 Relationship among Meta Model, Certification Model, and Certification Program

For easier explanation about a relationship among these, we use features of C++, a representative objective-oriented language (Figure 2). Just as a class template in C++ is a higher framework for generating classes, so the meta model is a higher framework for generating certification models. The meta model itself is not used directly for software certification, that is, the meta model is a reference model for generating certification models. Similarly, just as a class in C++ is a framework for generating objects, so a certification model is a framework for generating certification programs.



**Fig. 2.** Relationship among meta model, certification model, and certification program.



**Fig. 3.** Generation of certification model from meta model.

### 3.3 Applying Meta Model

The process of applying the meta model is largely divided into two steps: the generation of a certification model and a certification program.

#### Generation of Certification Model

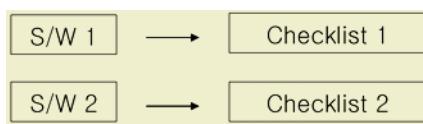
Just like generating various classes by using a class template in C++, it is possible to generate various certification models by assigning a standard or method to each element of the meta model. For example, we can use TTAS.KO-11.0026 [24] of Telecommunication Technology Association (TTA) as software domains, the characteristics of ISO/IEC 9126 [5] as quality factors, and ISO/IEC 12182 [21] as software environments (Figure 3).

#### Generation of Certification Program

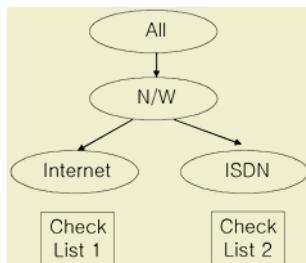
After generating a certification model, we can generate certification programs. At this step, we determine metrics, specific to software or software groups, for describing quality factors and then make checklists including the metrics. We identify three possible methods to generate certification programs: ad hoc approach, software group based approach, and software group based approach with specialization

In an ad hoc approach, whenever we certify a software product, we have to make checklists for the product. This approach is simple but inefficient because we must repeat making checklists even though a new software product belongs to the same group as a previously certified product belongs to (Figure 4).

In a software group based approach, we classify all kinds of software into groups hierarchically and make checklists for each group. When we certify a software product, we identify a group that the product belongs to and use the checklists of the group (Figure 5). This approach, however, has a disadvantage that though subgroups of a group have many common metrics, each lower group holds the checklists that redundantly have the common metrics.

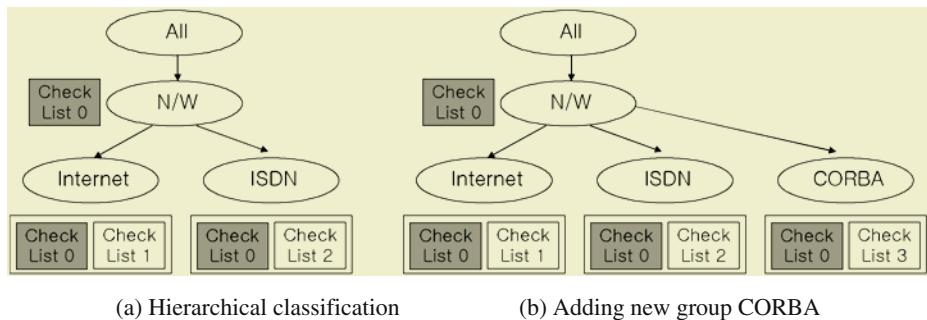


**Fig. 4.** Ad hoc approach.

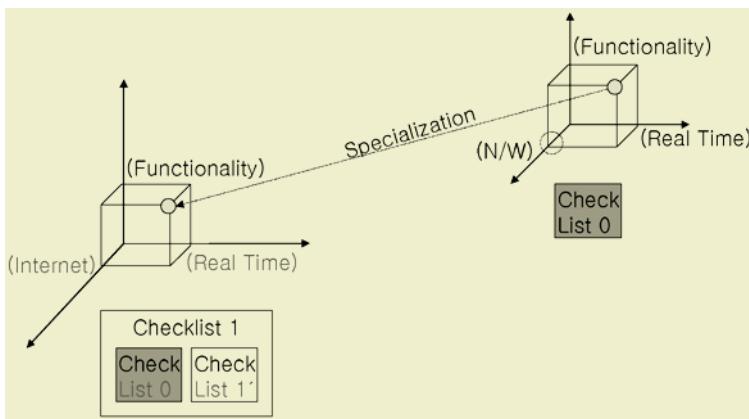


**Fig. 5.** Software group based approach.

A software group based approach with specialization is based on the software group based approach but has some differences. This approach works as follows. First, we make general checklists for higher groups in a software hierarchy and specific check-lists not including the general checklists for their lower groups. Next, we add the general checklists of higher groups to the checklists of each lower group. Finally, we define the added checklists as the checklists of the lower groups. For example, as shown in Figure 6 (a), the checklist of Internet group can be made by adding the checklist of its higher group N/W (Checklist 0) to the checklist specific to Internet (Checklist 1). This approach has an advantage that the checklists of new software groups can be constructed easily and incrementally. As shown in Figure 6 (b), new group CORBA can be added easily to the classification system without changing the current system. Therefore we use this approach in the meta model.



**Fig. 6.** Software group based approach with specialization.



**Fig. 7.** Specialization.

In this paper, we propose a way to incrementally construct checklists of each group from the root of a software hierarchy downward with making quality requirements of the group more rigorous than those of its higher groups, which is called specialization. For example, as shown in Figure 7, Checklist 1 can be created by adding the

checklist of its higher group (Checklist 0) to the specific checklist of its own group (Checklist 1'). Checklist 1 specifies evaluation techniques specifically created for software group Internet, for real-time environments, and for quality factor functionality.

**Table 1.** Structure of Checklist 0 in Fig. 7.

Metric	Critical value	Property
M1	A	Necessary
M2	B	Optional
M3	C	Necessary

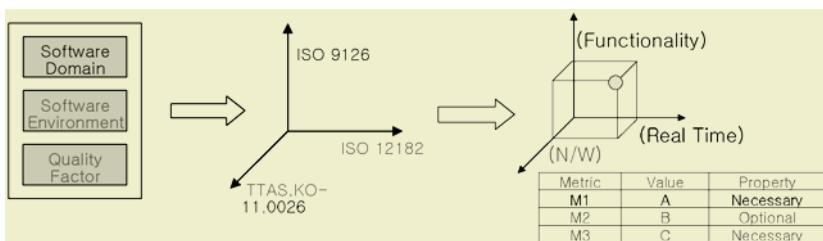
**Table 2.** Structure of Checklist 1 in Fig. 7.

Metric	Critical value	Property
M1	<u>D</u>	Necessary
M2	B	<u>Necessary</u>
M3	C	Necessary
<i>M4</i>	<i>E</i>	<i>Optional</i>
<i>M5</i>	<i>F</i>	<i>Optional</i>

Table 1 shows an example of a checklist. The checklist consists of metrics, values, and properties. However, checklists need not to have all the three elements. If necessary, certification organizations can omit the elements and add new ones. A critical value of each metric is a threshold value of the metric that is used to either accept or reject software in terms of the metric. A property of each metric informs whether the metric is used for certification necessarily or optionally.

Table 1 and 2 show that new metrics, M4 and M5, are added to Checklist 0 of the higher group in order to construct Checklist 1 of its lower group. We can also see that the value of M1 and the property of M2 changed in Checklist 1. It shows that quality requirements of a group can be more rigorous than those of its higher group. It is similar to overriding that is the concept used in Smalltalk and C++.

Figure 8 shows the relationship among the meta model, certification model, and certification program. The meta model is not a model used directly for software certification but a framework for generating certification models. A certification model is constructed by applying elements to the framework. Also with this certification model, a certification program is generated and finally the certification program is practically used for software certification.



**Fig. 8.** Relationship among meta model, certification model, and certification program.

## 4 Validation

In this section, we validate the meta model by generating certification programs with certification data on software packages which are certified by a third-party independent testing laboratory, Telecommunication Technology Association (TTA) in Korea.

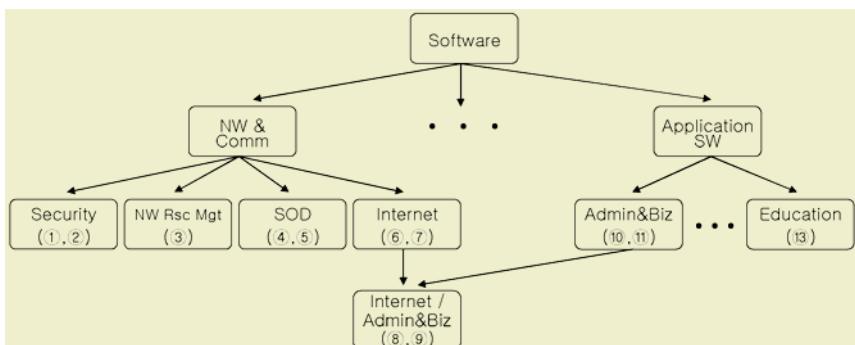
### 4.1 Generation of Certification Model

We adopt TTAS.KO-11.0026 of TTA [24] as software domains. The reasons are that there is not an international standard for software classification and that certification organizations generally have their own classification schemes.

As described in Section 3, software domains should be able to classify all kinds of software, have hierarchical structures, and be extensible. TTAS.KO-11.0026 of TTA satisfies these requirements. The characteristics of ISO/IEC 9126 are adopted as quality factors, because most testing laboratories follow this standard for quality evaluation and certification. ISO/IEC 12182 is referred to for software environments because it provides views that can describe environments in which software is operated.

### 4.2 Generation of Certification Program

In Section 3, we described three possible approaches to generation of certification programs. Among them, we use the software group based approach with specialization. Note that this approach can be usable under a prerequisite that certification programs are already produced for software groups. For this reason, we need to construct checklists for software groups. For the construction, we can use a deductive or an inductive method. In a deductive method, metrics for each software group may be determined without referring to quality information on certified products of the group. For example, an expert having abundant knowledge of a specific software group can construct a certification program in this way. On the other hand, in an inductive method, a certification program is constructed with quality data on certified software empirically. In other words, we can select common metrics specific to each software group by analyzing checklists of certified products of the group.



**Fig. 9.** Selected software packages and classification scheme.

**Table 3.** Checklists of 13 software packages.

Characteristic	Subcharacteristic	Metric	Network & Communication						App. SW			
			Security	NW	SOD	Internet			Admin& Biz	Fin	Edu	
			(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
Functionality	Suitability	Functional adequacy	●	●					●	●		●
	Suitability	Functional implementation completeness	●	●	●	●	●	●	●	●	●	●
	Suitability	Functional implementation coverage		●					●	●		●
	Suitability	Functional specification stability		●	●	●		●	●	●	●	●
	Accuracy	Computational accuracy	●	●	●	●	●	●	●	●	●	●
	Interoperability	Data exchangeability		●	●			●	●	●	●	●
	Security	Access controllability	●	●		●	●			●	●	●
	Security	Access auditability	●	●		●	●				●	
	Compliance	Functional compliance			●						●	●

**Table 4.** Checklist of Network & Communication group (●: necessary, ○: optional).

Characteristic	Subcharacteristic	Metric	Property
Functionality	Suitability	Functional adequacy	○
Functionality	Suitability	Functional implementation completeness	●
Functionality	Suitability	Functional specification stability	○
Functionality	Accuracy	Computational accuracy	●
Functionality	Interoperability	Data exchangeability	○
Functionality	Security	Access controllability	○
Reliability	Maturity	Failure resolution	○
Reliability	Maturity	Fault removal	○
Reliability	Fault tolerance	Breakdown avoidance	○
Reliability	Fault tolerance	Failure avoidance	○
Reliability	Fault tolerance	Incorrect operation avoidance	●
Reliability	Recoverability	Restore effectiveness	○
Efficiency	Time behavior	Mean time to response	○
Efficiency	Memory	Maximum memory utilization	○
Usability	Understandability	Functional understandability	●
Usability	Understandability	Demonstration Accessibility in use	●
Usability	Understandability	Understandable input and output	○
Usability	Understandability	Evident function	●
Usability	Understandability	Demonstration Accessibility	●
Usability	Understandability	Completeness of description	○
Usability	Learnability	Ease of function learning	●
Usability	Learnability	Help accessibility	○
Usability	Operability	Error correction	●
Usability	Operability	Message understandability in use	●
Usability	Operability	Operational consistency in use	●
Usability	Attractiveness	Interface appearance customizability	○
Usability	Attractiveness	Attractive interaction	●
Maintainability	Analyzability	Failure analysis capability	●
Maintainability	Changeability	Modification complexity	○
Portability	Adaptability	Adaptability of data structure	○
Portability	Adaptability	System software environment adaptability	●
Portability	Installability	Ease of installation	●
Portability	Installability	Ease of setup retry	●
Portability	Replaceability	User support functional consistency	○
Portability	Co-existence	Available co-existence	●

In this paper, an inductive method is applied. The method works as follows. First, we need to collect certified software products. We use thirteen software packages certified by TTA. Next, the thirteen software products need to be classified by TTAS.KO-11.0026 of TTA. For clear classification of the software, the standard is slightly modified. Figure 9 shows the software products (①~⑬) and the software classification system. Then, we need to collect the information about metrics used for quality evaluation. Table 3 partially shows the metrics. Out of external metrics of ISO/IEC 9126-2, we mark the metrics that TTA selected to certify each software product, with symbol ●. Finally, based on Table 3, the checklists of the principal software groups are derived as follows.

### **Network and Communication Group**

Network and Communication group is divided into Security (①, ②), Network Resource Management (③), SOD (Service On Demand) (④, ⑤), and Internet (⑥, ⑦, ⑧, ⑨) subgroups. Thus, metrics common to all the lower groups can be considered as metrics of Network and Communication group. In this work, we select metrics common to all the nine software products as metrics of Network and Communication group and mark their properties with necessary. Otherwise, if more than two groups out of the four lower groups have common metrics, we select the metrics as the metrics of the higher group and mark their properties with optional, as shown in Table 4. In this way, metrics of Application Software group can be determined.

### **Security Group**

We have to choose the metrics that are only involved with Security group and not involved with its higher group, Network and Communication. Metrics common to both of the two products (①, ②) are marked with necessary but metrics involved with only one product are marked with optional, as shown in Table 5. In the same way, metrics of the two groups, SOD and Network Resource Management, can be determined.

### **Internet Group**

The four products (⑥, ⑦, ⑧, ⑨) belong to Internet group. The two products among them (⑧, ⑨) inherit the characteristics of Administration and Business group. We choose metrics that are only involved with Internet group and not involved with Network and Communication group. Metrics common to all the four products are marked with necessary. In the work, all the metrics selected are optional, as shown in Table 6.

### **Administration and Business Group**

The two products (⑩, ⑪) belong to Administration and Business group. We choose metrics that are only involved with Administration and Business group and not involved with Application Software group, as shown in Table 7.

**Table 5.** Checklist of Security group (●: necessary, ○: optional).

Characteristic	Subcharacteristic	Metric	Property
Functionality	Security	Access auditability	●
Functionality	Compliance	Functional compliance	○
Reliability	Recoverability	Restorability	○
Efficiency	Time behavior	Mean time for turnaround	○
Maintainability	Analyzability	Diagnostic function support	○

**Table 6.** Checklist of Internet group (●: necessary, ○: optional).

Characteristic	Subcharacteristic	Metric	Property
Functionality	Suitability	Functional implementation coverage	○
Reliability	Maturity	Mean time between failure	○
Reliability	Recoverability	Availability	○
Reliability	Recoverability	Mean recovery time	○
Efficiency	Time behavior	Mean amount of throughput	○
Efficiency	Resource utilization	I/O device utilization	○
Portability	Adaptability	Porting user friendliness	○
Portability	Replaceability	Continued use of data	○

**Table 7.** Checklist of Administration & Business group (●: necessary, ○: optional).

Characteristic	Subcharacteristic	Metric	Property
Functionality	Suitability	Functional implementation coverage	○
Functionality	Security	Access auditability	○
Efficiency	Resource utilization	I/O device utilization	○
Usability	Operability	Operational error recoverability in use	○

## Internet/Administration and Business Group

This group including the two products (⑧, ⑨) multiply inherits from Internet group and Administration and Business group. So the final checklist of this group has to include not only metrics of Network and Communication group and Internet group, but also those of Administration and Business group. If we select metrics of the Internet/Administration and Business' own group, the metrics must not be involved with any groups above. There is no such metric in this work.

### 4.3 Discussion

In Subsection 4.2, we constructed the checklists of the software groups using quality data on the thirteen COTS software. In this subsection, we discuss problems identified during the construction.

The first problem is a problem of the reliability of checklists obtained by using the inductive method. When selecting metrics of a software group, the more the number of software products of the group is, the more reliable the metrics selected can be. From this point of view, the metrics of Network and Communication group selected from the nine products can be more reliable than those of the other groups. This problem may be solved if we have quality data on more certified software products.

Second, we need to establish a valid criterion for determining common metrics. If a metric is common to all subgroups of group  $g$ , we can consider the metric to be a metric of group  $g$  and then mark its property with necessary or optional. However, if a metric is common to some of the subgroups, we have to decide whether the metric is an optional metric of group  $g$  or the metric is an optional or necessary metric of each of the subgroups. To solve this problem, we need subject matter experts, statistical analysis, or artificial intelligence techniques such as neural networks.

## 5 Conclusion

Various types of COTS software are produced radically and its application areas are also being expanded. Because software quality evaluation depends on types of software and environments where software is operated, certification organizations need different certification programs that are foundations or basic frames for certifying software. In this paper, we have proposed a meta model for systematically generating and maintaining certification programs of software products. The certification meta model is a model which can be referred to by certification organizations to describe certification models including certification policies. By using the model, certification programs are incrementally constructed based on hierarchical classification of software. We have made a certification model by assigning a standard or method to each element of the meta model. To validate the meta model, we made prototype-level certification programs using quality data on software packages which are certified by a third-party independent testing laboratory, Telecommunication Technology Association (TTA) in Korea.

Our approach has a limit that it may not be easy to determine metrics specific to software groups or to decide certification criteria such as critical values of the metrics because the selection or decision directly affects certification results. Thus we need the large amount of data on certified software to make reliable checklists by using an inductive method.

In future work, we plan to develop tools that support the meta model, or methods that automatically generate certification programs through artificial intelligence techniques such as neural networks

## References

1. Jeffrey Voas, "Limited software warranties," In Proceedings of the 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 56-61, 2000.
2. Osman Balci, "A methodology for certification of modeling and simulation applications," ACM Transactions on Modeling and Computer Simulation, Vol. 11, No. 4, pp. 352-377, 2001.

3. Anca I. Vermesan, "Software certification for industry – verification and validation issues in expert systems," In Proceedings of the 9th International Workshop on Database and Expert Systems Applications, pp. 3-14, 1998.
4. ISO/IEC, "FCD 9126-1.2 Information Technology – Software product quality – Part 1: Quality model," ISO/IEC JTC1/SC7 N1949, 1998.
5. ISO/IEC, "DTR 9126-2: Software Engineering – Product Quality Part 2 – External Metrics," ISO/IEC JTC1/SC7 N2419, 2001.
6. ISO/IEC, "DTR 9126-3: Software Engineering – Product Quality Part 3 – Internal Metrics," ISO/IEC JTC1/SC7 N2416, 2001.
7. ISO/IEC, "DTR 9126-4: Software Engineering – Software product quality – Part 4: Quality In Use Metrics," ISO/IEC JTC1/SC7 N2430, 2001.
8. Dieter Welzel and Hans-Ludwig Hausen, "Practical Concurrent Software Evaluation for Certification," Journal of Systems and Software, Vol. 38, No. 1, pp. 71-83, 1997.
9. Hareton K. N. Leung, "Quality metrics for intranet applications," Information & Management, Vol. 38, No. 3, pp. 137-152, 2001.
10. Nigel Bevan, "Measuring usability as quality of use," Software Quality Journal, Vol. 4, pp. 115-150, 1995.
11. Motoei Azuma, "Software products evaluation system: quality models, metrics and processes – International Standards and Japanese Practice," Information and Software Technology, Vol. 38, No. 3, pp. 145-154, 1996.
12. ISO/IEC, "Information technology – Software product evaluation – Part 1: General overview," ISO/IEC 14598-1:1999(E), 1999.
13. ISO/IEC, "Software engineering – Product evaluation – Part 2: Planning and management," ISO/IEC 14598-2:2000(E), 2000.
14. ISO/IEC, "Software engineering – Product evaluation – Part 3: Process for developers," ISO/IEC 14598-3:2000(E), 2000.
15. ISO/IEC, "Software engineering – Product evaluation – Part 4: Process for acquirers," ISO/IEC 14598-4:1999(E), 1999.
16. ISO/IEC, "Information technology – Software product evaluation – Part 5: Process for evaluators," ISO/IEC 14598-5:1998(E), 1998.
17. ISO/IEC, "Software engineering – Product evaluation – Part 6: Documentation of evaluation modules," ISO/IEC 14598-6:1999(E), 1999.
18. Teade Punter, Rini van Solingen, and Jos Trienekens, "Software Product Evaluation – Current status and future needs for customers and industry," In the Proceedings of the 4th IT Evaluation Conference (EVIT-97), 1997.
19. Hans-Ludwig Hausen and Dieter Welzel, "Guides to Software Evaluation," Arbeitspapiere der GMD, No. 746, GMD, Sankt Augustin, April 1993.
20. ISO/IEC, "Information technology – Software packages – Quality requirements and testing," ISO/IEC 12119:1994(E), 1994.
21. ISO/IEC, "FCD 12182 Information Technology – Categorization of Software," ISO/IEC TR 12182:1998(E), 1998.
22. Jeffrey S. Poulin and Kathryn P. Yglesias, "Organization and Component Classification in the IBM Reuse Library," IBM Technical Report, TR 00.3730, 15 April 1993.
23. W. B. Frakes and T. P. Pole, "An Empirical Study of Representation Methods for Reusable Software Components," IEEE Transactions on Software Engineering, Vol. 20, No. 8, pp. 617-630, 1994.
24. Telecommunications Technology Association (TTA), "Standard for a Classification Scheme of Software Reuse," TTAS.KO-11.0026, 2000.

# An Automatic Test Data Generation System Based on the Integrated Classification-Tree Methodology<sup>\*,\*\*</sup>

Andrew Cain<sup>1</sup>, Tsong Yueh Chen<sup>1</sup>, Doug Grant<sup>1</sup>, Pak-Lok Poon<sup>2</sup>,  
Sau-Fun Tang<sup>1,3</sup>, and T.H. Tse<sup>4</sup>

<sup>1</sup> School of Information Technology  
Swinburne University of Technology  
Hawthorn 3122, Australia

<sup>2</sup> School of Accounting and Finance  
The Hong Kong Polytechnic University  
Hung Hom, Kowloon, Hong Kong  
afplpoon@inet.polyu.edu.hk

Phone: (+852) 2766 7072, Fax: (+852) 2356 9550

<sup>3</sup> Department of Finance and Decision Sciences  
Hong Kong Baptist University  
Kowloon Tong, Kowloon, Hong Kong

<sup>4</sup> Department of Computer Science and Information Systems  
The University of Hong Kong  
Pokfulam Road, Hong Kong

**Abstract.** Grochtmann and Grimm have developed the classification-tree method (CTM) to facilitate software testers in generating test cases from functional specifications. While the method is very useful, it is hindered by the lack of a systematic tree construction algorithm. This problem has been alleviated by Chen et al. via their “integrated” classification-tree methodology (ICTM). In this paper, we describe and discuss a prototype system ADDICT that is built on ICTM.

**Keywords:** Automatic test case generation, black box testing, category-partition method, choice relation framework, classification-tree method, software testing

## 1 Introduction

The generation of test cases is an important aspect in software testing because it affects the scope and, hence, the quality of the process [1,7]. This importance has inspired researchers to develop various test case generation methods. Among these researchers, Grochtmann and Grimm [8] have developed the *classification-tree method* (CTM). The major concept of the method is to generate test cases via the construction of classification trees (which we shall denote by  $T_s$ ). Although the concept of CTM is promising, this

\* A preliminary version of this paper was presented at the 1st ACIS International Conference on Software Engineering Research and Applications (SERA '03) [3].

\*\* This work is supported in part by a grant of the Research Grants Council of Hong Kong (Project No. HKU 7029/01E), a research and conference grant of The University of Hong Kong, and a grant from the Australian Research Council (ARC Discovery Project: DP 0345147).

method has a major weakness—the absence of a systematic tree construction algorithm. As a result, users of CTM are left with a loosely defined task of constructing  $\mathcal{T}$ s. For complicated specifications, this construction task could be difficult and hence prone to human errors. If a  $\mathcal{T}$  is wrongly constructed, the quality of the resulting test cases generated from it will be adversely affected.

The problem of the absence of a tree construction algorithm is alleviated by Chen et al. via their integrated classification-tree methodology (ICTM) [5]. With this methodology, software testers can construct  $\mathcal{T}$ s by using a systematic tree construction algorithm. In this paper, we discuss the development and functionality of a prototype system ADDICT (which stands for **A**utomate**D** test **D**ata generation using the **I**ntegrated **C**lassification-**T**ree methodology) built upon ICTM.

The rest of the paper is organized as follows. Section 2 outlines the major concept of ICTM [5]. Section 3 describes in detail the hardware and software platforms on which ADDICT is built, the various system features of ADDICT, and the major contribution of ADDICT. Section 4 discusses our planned extension to ADDICT. Section 5 describe other work related to CTM and ICTM. Finally, Sect. 6 summarizes and concludes the paper.

## 2 Overview of the Integrated Classification-Tree Methodology (ICTM)

Basically, ICTM [5] helps testers generate test cases from specifications via the construction of  $\mathcal{T}$ s. The tree construction task is supported by a predefined algorithm. ICTM consists of the following steps:

- (1) Decompose the specification into several *functional units*  $\mathcal{U}$ s that can be tested independently. For each  $\mathcal{U}$  selected for testing, repeat steps (2) to (7) below.
- (2) Identify classifications and their associated classes for the selected  $\mathcal{U}$ . *Classifications* are different criteria for partitioning the input domain of the selected  $\mathcal{U}$ , whereas *classes* are disjoint subsets of values for each classification. For every classification  $[X]$ , its associated classes should partition the possible values of  $[X]$  completely<sup>1</sup>. The grouping of certain values in a single class  $|X: x|$  indicates the belief that a test case with any value in  $|X: x|$  is essentially as good as one with any other value in that class [11].
- (3) Construct a *classification-hierarchy table*  $\mathcal{H}_{\mathcal{U}}$  for  $\mathcal{U}$ , which captures the hierarchical relation for each pair of classifications.
- (4) Construct a *classification tree*  $\mathcal{T}_{\mathcal{U}}$  from  $\mathcal{H}_{\mathcal{U}}$ .
- (5) Construct a *combination table* from  $\mathcal{T}_{\mathcal{U}}$ . Various combinations of classes can then be selected from the table according to a set of predefined selection rules. Each of these combinations of classes is called a *potential test frame*  $B$ .
- (6) Check all  $B$ 's against  $\mathcal{U}$ , with a view to identifying whether they are complete or incomplete. Given a  $B$ , if a standalone input to  $\mathcal{U}$  can be formed by selecting one element from every class in  $B$ , then  $B$  is a *complete test frame* (denoted by  $B^c$ ).

---

<sup>1</sup> In this paper, classifications are enclosed by square brackets [ ] while classes are enclosed by vertical bars | |. Furthermore, the notation  $|X: x|$  denotes class  $x$  in classification  $X$ .

Otherwise,  $B$  is *incomplete*. Incomplete test frames are not useful to testing and, hence, should be discarded before testing commences.

- (7) From each  $B^c$ , construct a test case by selecting one element from each class in  $B^c$ .

In step (6) above, two potential reasons for a  $B$  to be incomplete are: (a)  $B$  contains insufficient classes to form a standalone input to  $\mathcal{U}$ , and/or (b) the combination of classes in  $B$  contradicts  $\mathcal{U}$ . Readers may refer to [5] for details. Nevertheless, when we describe the various system features of ADDICT in Sect. 3.2, we shall elaborate on the above steps with examples.

## 3 ADDICT: A Prototype System for Automated Test Case Generation

### 3.1 Technical Details

We have carefully considered the hardware and software platforms on which ADDICT should be built. To improve its applicability, we have implemented ADDICT on the standard PC platform with Microsoft Windows as the operating system. ADDICT is written in Delphi, which is a Pascal-based object-oriented programming language. We have applied object-oriented techniques when designing and coding ADDICT. For example, a classification  $[X]$  is an aggregation of classes  $|X: x_1|, |X: x_2|, \dots, |X: x_k|$ , and  $[X]$  is related to other classifications through hierarchical relationships. In general, ADDICT does not impose a maximum limit on the number of classifications and classes, as long as the available memory in the PC can support them. The current version of ADDICT supports steps (2) to (5) of ICTM outlined in Sect. 2 above.

### 3.2 Functionality of ADDICT

We use a commercial specification, denoted by PURCHASE, to explain each step of ICTM mentioned in Sect. 2 and to describe the various features of ADDICT. Part of the specification is listed below:

#### **Part of the Specification PURCHASE for the Program $\mathcal{P}_{\text{PURCHASE}}$ :**

XYZ is an international bank that issues credit cards to approved customers. ... For each purchase,  $\mathcal{P}_{\text{PURCHASE}}$  shall accept the transaction details together with the various information of the credit card. Thereafter, validation of these details is performed in order to determine whether the purchase should be approved. The following are the various inputs to  $\mathcal{P}_{\text{PURCHASE}}$ :

##### **(a) Details of Credit Cards:**

- **Class of Credit Card:** Either “Gold” or “Classic”.
- **Credit Limit of Credit Card:** For gold credit cards, the credit limit is either \$5 000 or \$6 000. For classic credit cards, the credit limit is either \$2 000 or \$3 000.
- ...

**(b) Details of Purchase:**

- **Current Purchase Amount:** It can be any amount greater than \$0.00.
  - ...
- 

**Step (1) of ICTM (Decomposition of Specification):**

The first step is to decompose PURCHASE into a number of independent functional units  $\mathcal{U}$ s. In our case, because of the simplicity of PURCHASE, no decomposition is needed. In other words, the entire specification can be treated as one functional unit denoted by  $\mathcal{U}_{\text{PURCHASE}}$ .

**Step (2) of ICTM (Identification of Classifications and Classes):**

From  $\mathcal{U}_{\text{PURCHASE}}$ , the tester identifies 9 classifications and 22 classes. The number of classes contained in a classification ranges from 2 to 5. The following lists four examples of these classifications together with their associated classes:

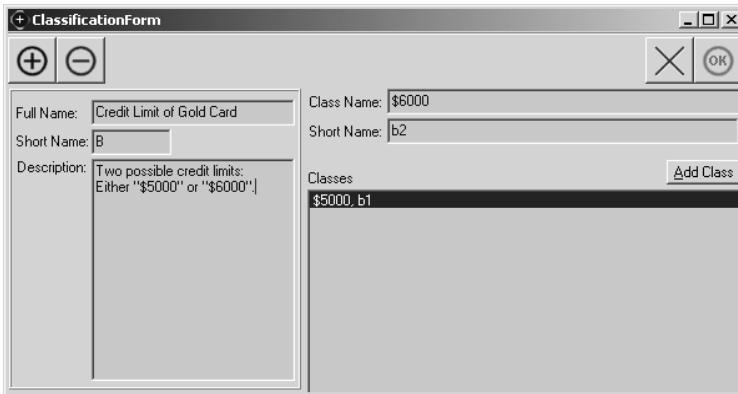
- Classification [Class of Credit Card], with |Class of Credit Card: Gold| and |Class of Credit Card: Classic| as its two associated classes.
- Classification [Credit Limit of Gold Card], with |Credit Limit of Gold Card: \$5,000| and |Credit Limit of Gold Card: \$6,000| as its two associated classes.
- Classification [Credit Limit of Classic Card], with |Credit Limit of Classic Card: \$2,000| and |Credit Limit of Classic Card: \$3,000| as its two associated classes.
- Classification [Current Purchase Amount ( $PA$ )], with |Current Purchase Amount:  $PA \leq \$2\,000.00$ |, |Current Purchase Amount:  $\$2\,000.00 < PA \leq \$3\,000.00$ |, |Current Purchase Amount:  $\$3\,000.00 < PA \leq \$5\,000.00$ |, |Current Purchase Amount:  $\$5\,000.00 < PA \leq \$6\,000.00$ |, and |Current Purchase Amount:  $PA > \$6\,000.00$ | as its five associated classes.

It can be seen from (d) above that a class can be defined for a range of values and, hence, although all the classes in a classification  $[X]$  should cover the input domain relevant to  $[X]$ , the number of classes in  $[X]$  is not necessarily large.

Consider Fig. 1 which depicts an input screen provided by ADDICT for entering the full and short names of classifications and classes. In this figure, the tester has defined classification [Credit Limit of Gold Card] in the upper-left box, and class |Credit Limit of Gold Card: \$5 000| in the bottom-right box. Additionally, the tester is adding class |Credit Limit of Gold Card: \$6 000| through the upper-right box.

**Step (3) of ICTM (Construction of Classification-Hierarchy Table):**

After entering all the classifications and classes into ADDICT, the next step is to construct the classification-hierarchy table  $\mathcal{H}_{\text{PURCHASE}}$  for PURCHASE. Here, the tester defines the hierarchical relation for each pair of classifications  $[X]$  and  $[Y]$  (denoted by  $[X] \rightarrow [Y]$ ). There are four possible types of hierarchical relation as follows:



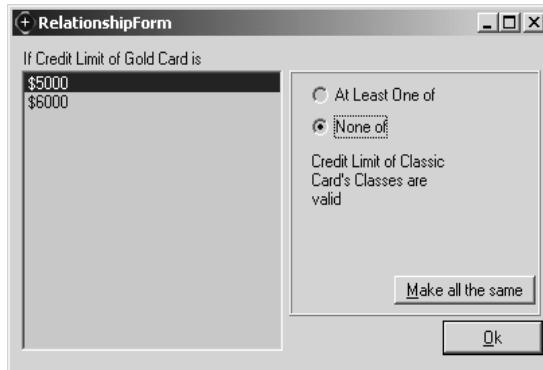
**Fig. 1.** Input screen for classifications and classes

- $[X]$  is a *loose ancestor* of  $[Y]$  (denoted by  $[X] \Leftrightarrow [Y]$ ),
- $[X]$  is a *strict ancestor* of  $[Y]$  (denoted by  $[X] \Rightarrow [Y]$ ),
- $[X]$  is *incompatible with*  $[Y]$  (denoted by  $[X] \sim [Y]$ ), and
- $[X]$  has *other relations* with  $[Y]$  (denoted by  $[X] \otimes [Y]$ ).

In the above, the symbols “ $\Leftrightarrow$ ”, “ $\Rightarrow$ ”, “ $\sim$ ”, and “ $\otimes$ ” are known as *hierarchical operators*. Readers may refer to [5] for details, especially the conditions in determining the correct hierarchical relation for  $[X] \rightarrow [Y]$ . Note that the conditions associated with each of the above hierarchical relations are mutually exclusive and exhaustive and, hence,  $[X] \rightarrow [Y]$  is well defined. These hierarchical relations will determine the relative position of  $[X]$  and  $[Y]$  in  $\mathcal{T}$ . For example,  $[X] \Rightarrow [Y]$  corresponds to the situation where  $[X]$  will appear as either a parent or an ancestor of  $[Y]$  in  $\mathcal{T}$ <sup>2</sup>.

Figure 2 depicts an input screen to capture the constraints of [Credit Limit of Gold Card] on [Credit Limit of Classic Card]. These captured constraints will be used by ADDICT to determine the appropriate hierarchical operator for [Credit Limit of Gold Card]  $\rightarrow$  [Credit Limit of Classic Card]. In the input screen, the tester indicates that [Credit Limit of Gold Card: \$5 000] cannot be combined with any class (that is, [Credit Limit of Classic Card: \$2 000] and [Credit Limit of Classic Card: \$3 000]) in [Credit Limit of Classic Card] to form part of any complete test frame. By clicking the “Make all the same” button near the bottom-right part of the input screen, the tester also indicates that the constraint of [Credit Limit of Gold Card: \$6 000] on all the classes in [Credit Limit of Classic Card] is the same as that of [Credit Limit of Gold Card: \$5 000] on all the classes in [Credit Limit of Classic Card]. This saves the effort in defining all constraints individually. Based on the entered constraints in Fig. 2, ADDICT will automatically assign the hierarchical operator “ $\sim$ ” to [Credit Limit of Gold Card]  $\rightarrow$

<sup>2</sup> For the *parent-child* relation, a classification is “directly” placed under one or more classes of another classification. For the *ancestor-descendant* relation, a classification is “indirectly” placed under one or more classes of another classification.



**Fig. 2.** Input screen for constraints between a pair of classifications

[Credit Limit of Classic Card]. In short, ADDICT will determine and assign the appropriate hierarchical operator to  $[X] \rightarrow [Y]$ , based on the captured constraints of  $[X]$  on  $[Y]$ .

Figure 3 depicts the completed  $\mathcal{H}_{\text{PURCHASE}}$  with every element in it contains a hierarchical operator and corresponds to the hierarchical relation between a pair of classifications<sup>3</sup>. We use  $t_{ij}$  to denote the element in the  $i$ th row and the  $j$ th column of  $\mathcal{H}_{\mathcal{U}}$ . Consider, for example,  $t_{23}$  in  $\mathcal{H}_{\text{PURCHASE}}$ . It contains the hierarchical operator “~”, and corresponds to [Credit Limit of Gold Card] ~ [Credit Limit of Classic Card]. Note that the background color of all unassigned elements in  $\mathcal{H}_{\text{PURCHASE}}$  is initially set to “blue”. Once the constraints corresponding to an element  $t_{ij}$  have been entered and a hierarchical operator has been assigned to it, the background color of that element will change to “white”.

With regard to the construction of  $\mathcal{H}_{\mathcal{U}}$ , the following features provided by ADDICT are worth mentioning:

- (a) A constraint of ICTM is that the parent-child or ancestor-descendant hierarchical relation must be *anti-symmetric* for any pair of classifications. Otherwise a  $\mathcal{T}$  cannot be constructed. In other words,  $[X] \Rightarrow [Y]$  must imply  $[X] \not\Rightarrow [Y]$ . Software testers may need to redefine the original set of classifications and classes in order to meet this constraint while preserving the requirements of the target system (see [5] for details).

Regarding this issue, ICTM helps testers identify such unwarranted situations by means of the hierarchical operator “ $\Leftrightarrow$ ”. Whenever  $[X] \Leftrightarrow [Y]$  is being defined, we know that a symmetric parent-child or ancestor-descendant hierarchical relation occurs between  $[X]$  and  $[Y]$ . In this case, testers will be alerted to redefine  $[X]$  and  $[Y]$  (and their associated classes) so as to prevent a loop in  $\mathcal{T}$ .

<sup>3</sup> Note that, short names instead of full names for the classifications (both names are entered via the input screen as depicted in Fig. 1) are displayed as row and column headings in  $\mathcal{H}_{\text{PURCHASE}}$ . The idea is to fit the entire  $\mathcal{H}_{\text{PURCHASE}}$  into the screen. In the situation where  $\mathcal{H}_{\text{PURCHASE}}$  is too large (because of too many classifications) that exceeds the size of the screen, then vertical and horizontal scroll bars can be used to view different parts of  $\mathcal{H}_{\text{PURCHASE}}$ .

**Fig. 3.** Classification-hierarchy table  $\mathcal{H}_{\text{PURCHASE}}$  for PURCHASE

Consider  $t_{12}$  and  $t_{21}$  in  $\mathcal{H}_{\text{PURCHASE}}$  of Fig. 3. They correspond to ([Class of Credit Card]  $\Rightarrow$  [Credit Limit of Gold Card]) and ([Credit Limit of Gold Card]  $\otimes$  [Class of Credit Card]), respectively. Suppose, during the process of entering the constraints between these two classifications, the tester has made a mistake and eventually caused ADDICT to assign the hierarchical operator “ $\Leftrightarrow$ ” to both  $t_{12}$  and  $t_{21}$ . Accordingly, the background color of  $t_{12}$  and  $t_{21}$  will change from “white” to “red”, thus alerts the tester that symmetric parent-child or ancestor-descendant hierarchical relations occur. Note that, in this case, the unwarranted situation happens to occur because of an input error; symmetric parent-child or ancestor-descendant hierarchical relations in fact do not exist in  $\mathcal{U}_{\text{PURCHASE}}$ . In some other cases, however, this occurrence may result from correct inputs because symmetric parent-child or ancestor-descendant hierarchical relations do exist between some pairs of classifications identified from  $\mathcal{U}$ .

- (b) In [5], Chen et al. have identified three properties of the hierarchical relations as follows:

**Property 1:** If  $[X] \Rightarrow [Y]$ , then  $[Y] \otimes [X]$ .

**Property 2:** If  $[X] \sim [Y]$ , then  $[Y] \sim [X]$ .

**Property 3:** If  $[X] \otimes [Y]$ , then  $[Y] \Rightarrow [X]$  or  $[Y] \otimes [X]$ .

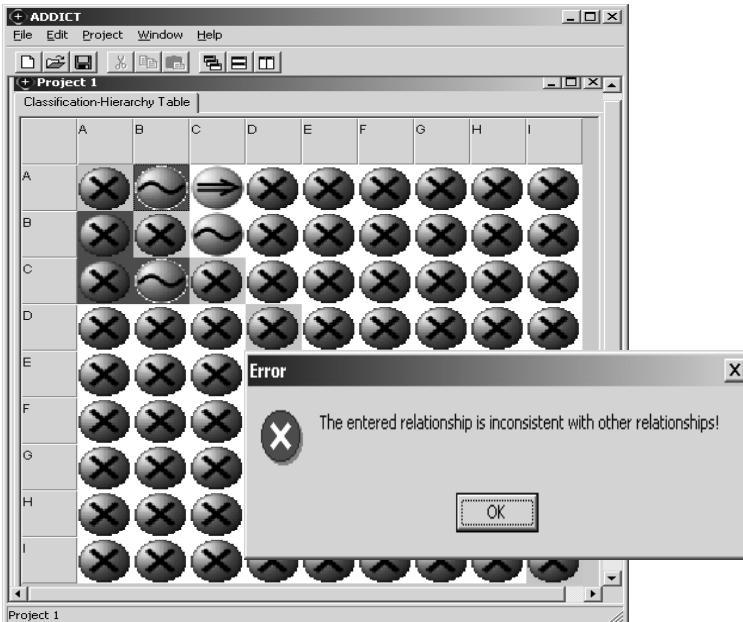
Using these properties, ADDICT provides a certain degree of automatic deduction and consistency check during the construction of  $\mathcal{H}_{\mathcal{U}}$ . Examples are given as below:

- (i) **Automatic Deduction:** Consider Fig. 2 again. This input screen is used to enter the constraints of each class in [Credit Limit of Gold Card] on [Credit Limit of Classic Card]. The entered constraints cause ADDICT to assign the hierarchical operator “ $\sim$ ” to [Credit Limit of Gold Card]  $\rightarrow$  [Credit Limit of Classic Card]. Later, without automatic deduction, the tester is required to enter the constraints of each class in [Credit Limit of Classic Card] on [Credit Limit of Gold Card] via another input screen similar to Fig. 2, if such constraints have not yet been entered. Now, by using Prop. 2, this requirement no longer exists because ADDICT will automatically deduce the hierarchical operator for [Credit Limit of Classic Card]  $\rightarrow$  [Credit Limit of Gold Card] to be “ $\sim$ ”. Accordingly, the background color for  $t_{32}$  (which corresponds to [Credit Limit of Classic Card]  $\sim$  [Credit Limit of Gold Card]) will change from “blue” to “green” to inform the tester that the hierarchical operator for  $t_{32}$  is automatically deduced (note that the background color for all the table elements whose hierarchical operators are manually defined is “white”). Besides Prop. 2, ADDICT will also provide automatic deduction based on Prop. 1. In fact, with the feature of automatic deduction, only about three-quarters of the hierarchical relations in  $\mathcal{H}_{\text{PURCHASE}}$  have to be manually defined.
- (ii) **Consistency Checking:** Consider  $t_{12}$  and  $t_{21}$  in  $\mathcal{H}_{\text{PURCHASE}}$  in Fig. 3 again, which correspond to ([Class of Credit Card]  $\Rightarrow$  [Credit Limit of Gold Card]) and ([Credit Limit of Gold Card]  $\otimes$  [Class of Credit Card]), respectively. Suppose,
- The constraints for  $t_{21}$  are entered before that for  $t_{12}$ .
  - The constraints for  $t_{21}$  are entered correctly, causing ADDICT to assign the hierarchical operator “ $\otimes$ ” to  $t_{21}$ .
  - Thereafter, the tester has made a mistake during the entry of the constraints for  $t_{12}$ , causing ADDICT to incorrectly assign the hierarchical operator “ $\sim$ ” to  $t_{12}$ .

This mistake is undesirable because incorrect hierarchical relations will eventually result in the generation of incomplete test frames, or the omission of some complete test frames. Regarding this problem, ADDICT provides a consistency check for the defined hierarchical relations. In fact, the incorrect hierarchical operator “ $\sim$ ” for  $t_{12}$  will be detected as an inconsistency by ADDICT with reference to Prop(s). (2) and (3) mentioned above. This is because the combination of ([Class of Credit Card]  $\sim$  [Credit Limit of Gold Card]) and ([Credit Limit of Gold Card]  $\otimes$  [Class of Credit Card]) contradicts these two properties. Accordingly, the background of  $t_{12}$  and  $t_{21}$  in  $\mathcal{H}_{\text{PURCHASE}}$  will change from “white” to “red” to alert the tester to take correction actions. An alert message box will also be displayed automatically by ADDICT to inform the tester about the inconsistency (see Fig. 4).

In summary, ADDICT adopts the following principles in order to improve on the effectiveness and efficiency of table construction:

- To perform automatic deduction instead of manual definition for each unassigned  $t_{ij}$  whenever possible.
- To perform consistency checking after every manual definition of  $t_{ij}$ .



**Fig. 4.** Message box to alert users about inconsistent hierarchical relations

#### Step (4) of ICTM (Construction of Classification Tree):

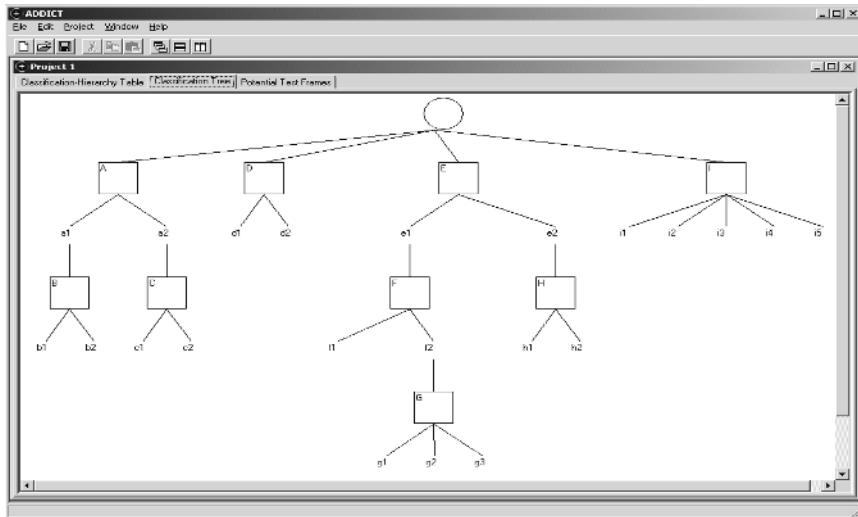
Based on the completed  $\mathcal{H}_{\text{PURCHASE}}$  in Fig. 3, ADDICT will automatically construct the corresponding classification tree  $\mathcal{T}_{\text{PURCHASE}}$  (see Fig. 5), based on a predefined tree construction algorithm provided in [5]. Similarly to  $\mathcal{H}_{\text{PURCHASE}}$ , short names are used for the classifications and classes in displaying  $\mathcal{T}_{\text{PURCHASE}}$ , and vertical and horizontal scroll bars can be used to view different parts of  $\mathcal{T}_{\text{PURCHASE}}$  if the tree is too large to fit into the screen.

In step (5) of ICTM described in Sect. 2, potential test frames  $B$ 's are generated by selecting combinations of classes from the combination table of  $\mathcal{T}$ , based on certain selection rules. Thereafter, the combination of classes in every  $B$  has to be checked against  $\mathcal{U}$ , with a view to classifying  $B$  either as a complete test frame  $B^c$  or as an incomplete test frame. The reason for checking is because, occasionally, a  $\mathcal{T}$  may not be able to capture all the constraints and relationships among classifications identified from  $\mathcal{U}$ . This problem results in the selection of some incomplete test frames from the combination table of  $\mathcal{T}$ .

Let  $N_B$  and  $N_{B^c}$  denote the total number of  $B$ 's and  $(B^c)$ 's, respectively, selected from the combination table of  $\mathcal{T}$ . In [5], Chen et al. define an effectiveness metric  $E_{\mathcal{T}}$  for any  $\mathcal{T}$  as:

$$E_{\mathcal{T}} = \frac{N_{B^c}}{N_B} \quad (1)$$

$E_{\mathcal{T}}$  is defined as such based on the argument that  $\mathcal{T}$  is merely a means to construct  $(B^c)$ 's for testing. The ideal situation is that all  $B$ 's are complete (that is,  $N_B = N_{B^c}$ ) and, hence,



**Fig. 5.** Classification tree  $\mathcal{T}_{\text{PURCHASE}}$  for PURCHASE

$E_{\mathcal{T}} = 1$ . Obviously, a small value of  $E_{\mathcal{T}}$  is undesirable since more effort is required to identify all the incomplete test frames. Furthermore, this manual identification process is prone to human errors, especially when  $N_B$  is large. If some  $(B^c)$ 's are somehow mistakenly classified as incomplete and hence not being used, the comprehensiveness of testing will be adversely affected.

Chen et al. [5] observe that a major reason for a small value of  $E_{\mathcal{T}}$  is the occurrence of duplicated subtrees in  $\mathcal{T}$ . To deal with this problem, they develop a classification tree restructuring technique to suppress the occurrence of duplicated subtrees in  $\mathcal{T}$ . This restructuring technique is part of their integrated classification tree construction algorithm. Two important properties of this restructuring technique are: (i) to reduce the value of  $N_B$  by pruning some duplicated subtrees from  $\mathcal{T}$ , and (ii) to retain all the  $(B^c)$ 's and, hence,  $N_{B^c}$  remains unchanged. Because of these two properties, the value of the effectiveness metric  $E_{\mathcal{T}}$  can be increased. Readers may refer to [5] for details.

In ADDICT, the construction of the resulting  $\mathcal{T}$  is performed on an incremental basis — classifications and classes are firstly assembled together to form subtrees, which in turn are joined together to form the resulting  $\mathcal{T}$ . During the tree construction process, ADDICT will automatically detect the occurrence of duplicated subtrees. If duplicated subtrees do exist, ADDICT will apply the tree restructuring technique by Chen et al., in order to increase the value of  $E_{\mathcal{T}}$  of the resulting  $\mathcal{T}$ . Note that the tree construction process, that incorporates the restructuring technique, is performed by ADDICT in a fully automatic manner without human intervention.

#### Step (5) of ICTM (Construction of Combination Table and Selection of Potential Test Frames):

With  $\mathcal{T}_{\text{PURCHASE}}$  in Fig. 5, the next step is to construct the corresponding combination table, from which  $B$ 's can be selected. This step is rather straightforward by following

some selection rules given in [5], which will not be repeated here. Same as  $\mathcal{T}_{\text{PURCHASE}}$ , the construction of the combination table and the selection of  $B$ 's are done by ADDICT automatically. In our case, a total of 240  $B$ 's will be selected from the combination table of  $\mathcal{T}_{\text{PURCHASE}}$  by ADDICT. Figure 6 shows a partial list of  $B$ 's constructed by ADDICT.

The screenshot shows the ADDICT software interface with a window titled 'Project 1'. The main area displays a table with 24 rows, each representing a potential test frame ('Frame 1' through 'Frame 24'). The columns represent various attributes and their values, such as 'Class of Credit Card' (e.g., Gold), 'Credit Limit of Gold Card' (\$6000), 'Type of Credit Card' (e.g., Corporate), 'Purchase of Airline Ticket' (e.g., Yes), and 'Airline Company' (e.g., ABC). The table is a 'Classification-Hierarchy Table' as indicated in the title bar.

Frame	Class of Credit Card	Credit Limit of Gold Card	Type of Credit Card	Purchase of Airline Ticket	Airline Company
Frame 1	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = ABC
Frame 2	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = ABC
Frame 3	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 4	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 5	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 6	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 7	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 8	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 9	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 10	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 11	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 12	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 13	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 14	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 15	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 16	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 17	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 18	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 19	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 20	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = Yes	Airline Company = Other Airlines
Frame 21	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = No	Type of Outlet = Bonus Outlet
Frame 22	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = No	Type of Outlet = Bonus Outlet
Frame 23	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = No	Type of Outlet = Bonus Outlet
Frame 24	Class of Credit Card = Gold	Credit Limit of Gold Card = \$6000	Type of Credit Card = Corporate	Purchase of Airline Ticket = No	Type of Outlet = Bonus Outlet

Fig. 6. Partial list of potential test frames for PURCHASE

### Step (6) of ICTM (Differentiation between Complete and Incomplete Test Frames):

As discussed in step (4) above, the tester has to check all the  $B$ 's with  $\mathcal{U}_{\text{PURCHASE}}$  to see whether any of them is incomplete. In our case, no incomplete test frame exists and, hence, all the 240  $B$ 's are also complete.

### Step (7) of ICTM (Construction of Test Cases):

For each of the 240 ( $B^c$ )'s, the tester selects an element from each class contained in  $B^c$  to form a test case. Consider, for example, the following  $B_1^c$  for  $\mathcal{U}_{\text{PURCHASE}}$  generated by ADDICT:

$$\{| \text{Class of Credit Card: Gold} |, | \text{Credit Limit of Gold Card: \$6 000} |, | \text{Current Purchase Amount (PA): \$5 000.00} < PA \leq \$6 000.00 |, \dots \}$$

A possible test case for  $B_1^c$  is:

$$(\text{Class of Credit Card} = \text{Gold}, \text{Credit Limit of Gold Card} = \$6 000, \text{Current Purchase Amount} = \$5 123.40, \dots)$$

Obviously, a total of 240 test cases will be constructed in this step for testing  $\mathcal{U}_{\text{PURCHASE}}$ .

### 3.3 Major Contribution of ADDICT

As mentioned earlier in the paper, the main purpose of ICTM and ADDICT is to provide a systematic method for the construction of  $\mathcal{T}$ 's from specifications. This feature does not

exist in CTM. Hence, users of CTM have to construct  $\mathcal{T}$ s in an ad hoc manner based on their knowledge and experience. This ad hoc approach does not provide assurance on the quality of the constructed  $\mathcal{T}$ s. If these  $\mathcal{T}$ s are incorrectly constructed, some ( $B^c$ )'s may not be generated and, hence, parts of the system that contain faults may not be tested. In this respect, the contribution of ICTM and ADDICT is obvious. Readers may note that the effectiveness of ICTM and ADDICT is greatly improved via the automatic detection of symmetric parent-child or ancestor-descendant hierarchical relations, the automatic deduction and consistency checking of hierarchical relations, and the automatic detection and removal of duplicated subtrees (in order to improve on the effectiveness metric  $\mathcal{E}_{\mathcal{T}}$ ).

## 4 Planned Extensions to ADDICT

The current version of ADDICT supports steps (2) to (5) of ICTM outlined in Sect. 2. For step (1), the decomposition of the specification into several  $\mathcal{U}$ s is not a trivial task that can be easily automated. Similarly, it is difficult, if not impossible, to automate step (6) regarding the differentiation between complete and incomplete test frames.

With regard to step (7), we plan to extend the system features provided by the current version of ADDICT in the following two ways:

- (a) After the automatic construction of the combination table and the selection of  $B$ 's in step (5), the next task is to construct a test case from every  $B$ . This task can be performed automatically by ADDICT by arbitrarily selecting one element from every class contained in  $B$ . Note that, in this approach, the generated test cases may contradict  $\mathcal{U}$  because the  $B$ 's have not been checked against  $\mathcal{U}$  to determine whether they are complete or incomplete. Hence, the tester has to check all the generated test cases against  $\mathcal{U}$  to see which of them are useful for testing.
- (b) In (a) above, only one test case is generated for each  $B$ . (Similarly, in the original ICTM, only one test case is generated for each  $B^c$ .) In our planned extension to ADDICT, the approach in (a) can be made more flexible so that one or more test cases can be generated for each  $B$ . This caters for the situation where the tester can afford to test  $\mathcal{U}$  with more test cases.

## 5 Related Work

Finally, we would like to compare CTM and ICTM with other related work, thus allowing readers to have a better grasp of the current state of research and practice:

- (1) Ostrand et al. [2,11] have developed the *category-partition method* (CPM) for generating test cases from specifications. The basic approach of CPM is very similar to that of CTM/ICTM—all of them aim at constructing a model of the constraints in the input domain so that combinations of compatible classes<sup>4</sup> can be generated and combinations of incompatible classes can be suppressed as far as possible. The main difference between CPM and CTM/ICTM is how the constraints among

---

<sup>4</sup> “Classes” in CTM/ICTM are known as “choices” in CPM.

classes are captured. While the former captures the constraints via the notion of a formal test specifications (which is a list of categories<sup>5</sup>, choices, and constraints in textural format), the latter capture these constraints by means of a classification tree  $\mathcal{T}$ . CPM has also been enhanced by Chen et al. [6] via the *choice relation framework* so that the test case generation process can be more systematic.

- (2) Singh et al. [12] have developed a technique to generate test cases from Z specifications by combining CTM with disjunctive normal forms (DNFs). In this technique, “high-level” test cases are first generated from the Z specification via the construction of a  $\mathcal{T}$ . These high-level test cases are then refined by generating a DNF for them. Also working on Z, Hierons et al. [9] have introduced an approach that extracts predicates from a Z specification and constructs a  $\mathcal{T}$  from these predicates, thus showing how the construction of a  $\mathcal{T}$  can be semi-automated based on a formal specification.

Readers may note that the work described in [9] and [12] mainly focuses on the application of CTM to Z specifications, whereas our work in this paper is more general, in the sense that our prototype system ADDICT does not impose any limitation on the type of specification, as long as a set of classifications and classes can be identified.

- (3) In [10], Lehmann and Wegener have described a classification-tree editor (CTE) known as CTE XL (eXtended Logics). This editor is used to solve some weaknesses they have identified in CTM. An example of such weaknesses is that CTM does not provide a feature to specify logical dependencies between classes. Hence, when selecting potential test frames from the combination table, software testers have to take care of the logical compatibility of the classes themselves.

The objective of our work is quite different from that of [10]. Our prototype ADDICT helps testers construct  $\mathcal{T}$ s from specifications. On the other hand, CTE XL is mainly a classification-tree “editor” rather than a “generator”, and requires testers to construct a  $\mathcal{T}$  by themselves, usually in an ad hoc manner.

- (4) At the initial stage of CTM/ICTM, software testers have to identify a set of classifications and classes. Owing to the absence of a systematic identification technique, this identification process is currently performed in an ad hoc approach. Chen et al. [4] argue that this approach does not provide reasonable assurance on the quality of the identified classifications and classes, and hence on the quality of the resulting test cases. They have performed case studies to find out the common mistakes made by software testers when they identify classifications and classes from specifications in such an ad hoc approach.

## 6 Summary and Conclusion

In this paper, we have introduced ICTM and outlined its major steps. This is followed by discussions of the technical details and system features of ADDICT, particularly the automatic detection of symmetric parent-child or ancestor-descendant hierarchical relations, the automatic deduction and consistency checking of hierarchical relations,

---

<sup>5</sup> “Classifications” in CTM/ICTM are known as “categories” in CPM.

and the automatic detection and removal of duplicated subtrees. We have also highlighted the major contribution of ADDICT and two possible areas for extending the current version of ADDICT in order to make it more useful. We believe that ICTM is a viable method for generating test cases from specifications, especially with the support of appropriate automated tools such as ADDICT. We plan to perform case studies to further investigate the contributions of ADDICT, especially with respect to the application of the system to the testing of real-life software.

## References

1. P. Ammann and J. Offutt. Using formal methods to derive test frames in category-partition testing. *Safety, Reliability, Fault Tolerance, Concurrency, and Real Time Security: Proceedings of the 9th Annual IEEE Conference on Computer Assurance (COMPASS '94)*, pages 69–79. Los Alamitos, California: IEEE Computer Society Press, 1994.
2. M. J. Balcer, W. M. Hasling, and T. J. Ostrand. Automatic generation of test scripts from formal test specifications. *Proceedings of the 3rd ACM Annual Symposium on Software Testing, Analysis, and Verification (TAV '89)*, pages 210–218. New York: ACM Press, 1989.
3. A. Cain, T.Y. Chen, D. Grant, P.-L. Poon, S.-F. Tang, and T. H. Tse. ADDICT: a prototype system for automated test data generation using the integrated classification-tree methodology. *Proceedings of the 1st ACIS International Conference on Software Engineering Research and Applications (SERA '03)*, pages 76–81. Mt. Pleasant, Michigan: International Association for Computer and Information Science, 2003.
4. T.Y. Chen, P.-L. Poon, S.-F. Tang, and T.H. Tse. An experimental analysis of the identification of categories and choices from specifications. *Proceedings of the 3rd ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2003)*, pages 99–106. Mt. Pleasant, Michigan: International Association for Computer and Information Science, 2002.
5. T. Y. Chen, P.-L. Poon, and T. H. Tse. An integrated classification-tree methodology for test case generation. *International Journal of Software Engineering and Knowledge Engineering*, 10(6):647–679, 2000.
6. T. Y. Chen, P.-L. Poon, and T. H. Tse. A choice relation framework for supporting category-partition test case generation. *IEEE Transactions on Software Engineering*, 29(7):577–593, 2003.
7. T. Chusho. Test data selection and quality estimation based on the concept of essential branches for path testing. *IEEE Transactions on Software Engineering*, 13(5):509–517, 1987.
8. M. Grochtmann and K. Grimm. Classification trees for partition testing. *Software Testing, Verification and Reliability*, 3(2):63–82, 1993.
9. R. M. Hierons, M. Harman, and H. Singh. Automatically generating information from a Z specification to support the classification tree method. Volume 2651 of *Lectures Notes in Computer Science*, pages 388–407. Berlin, Heidelberg: Springer-Verlag, 2003.
10. E. Lehmann and J. Wegener. Test case design by means of the CTE XL. *Proceedings of the 8th European International Conference on Software Testing, Analysis and Review (EuroSTAR 2000)*, 2000.
11. T.J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *Communications of the ACM*, 31(6):676–686, 1988.
12. H. Singh, M. Conrad, and S. Sadeghipour. Test case design based on Z and the classification-tree method. *Proceedings of the 1st IEEE International Conference on Formal Engineering Methods (ICFEM '97)*, pages 81–90. Los Alamitos, California: IEEE Computer Society Press, 1997.

# Normalizing Class Hierarchies Based on the Formal Concept Analysis

Suk-Hyung Hwang<sup>1</sup>, Sung-Hee Choi<sup>1</sup>, and Hae-Sool Yang<sup>2</sup>

<sup>1</sup> Division of Computer and Information Science, SunMoon University  
100 Kal-San-Ri, Tang-Jeong-Myon, A-San, Chung-Nam, 336-840 Korea  
[{shwang, shchoi}@sunmoon.ac.kr](mailto:{shwang, shchoi}@sunmoon.ac.kr)

<sup>2</sup> Graduate School of Venture, Hoseo University  
29-1 Se-Chul-Ri, Bae-Bang-Myon, A-San, Chung-Nam, 336-795 Korea  
[hsyang@office.hoseo.ac.kr](mailto:hsyang@office.hoseo.ac.kr)

**Abstract.** Class hierarchies often constitute the backbone of object-oriented software systems. Building “good” class hierarchies is a very important and common task, but such hierarchies are not so easy to build and evolve. Therefore, their construction and evolution are very important issues in component-based and object-oriented software engineering. In this paper, we present a normalized form of class hierarchy based on the concept lattice of formal concept analysis. Our approach provides the theoretical bases for the creation and evolution of well-defined object-oriented class hierarchy structures.

## 1 Introduction

One of the most important productions of object-oriented analysis and design methods is the class diagrams that represent the class hierarchies of the system under construction. The class hierarchy offers a way to share(factorize) properties(attributes and methods) between classes. The class hierarchies are a cornerstone of frameworks i.e. of adaptable and reusable object-oriented architectures and, therefore its quality is very important. Any kind of automated help in building, reorganizing or maintaining class hierarchies can thus be of interest and can have applications in several important research areas of object-oriented technology[1]–[4].

Casais[1] introduces global and incremental class hierarchy reorganization algorithm to restructure inheritance hierarchies to avoid explicit rejection of inherited properties, but his work emphasizes rather the reorganization of the class hierarchy of a particular object-oriented language than the maintenance of the class hierarchy according to change requests of the users.

Johnson and Opdyke[2] suggest the high-level refactoring techniques for class hierarchy over the object-oriented frameworks. They study class restructuring of classes related by composition and inheritance. Their transformation set includes the creation of an abstract superclass, subclassing, and refactoring to capture aggregations and components. Their refactorings specifically apply to source code that performs programs transformations, but not for designs in early analysis and design phases.

In other hand, Tokuda and Batory[3] provide a refactoring approach based on three kinds of design evolution:database schema transformations, design pattern microarchitectures, and hot-spot meta patterns. However, such refactorings are usually manipulate

portions of the system below the method level that references to program elements that are being changed. Bergstein[4] considers the object equivalence relationship between class hierarchies, and suggests a list of the object preserving primitive transformations to reorganize inheritance hierarchy structures. But, the order of the transformation operations is not considered.

Actually, these research works have provided the algorithms and heuristics to produce “good” and “reusable” class hierarchy organizations. However, common object-oriented methods do not give a satisfying answer to the question how to find the appropriate classes and objects to form the class hierarchy.

We propose here to adopt the concept lattice of Formal Concept Analysis as a framework for the design of class hierarchies because it is a natural structure that systematically factors out commonalities while preserving specialization relationships between classes. In fact, these correspond to the sets of initial classes and of all available properties, respectively. A formal concept in this model associates a (closed) set of objects and the set of attributes they share, whereas the set of all concepts forms a complete lattice when provided with set inclusion between object components. The concept lattice can be considered as a kind of normal form for the design of class hierarchies. The lattice shows all potentially useful generalizations.

The structure of the rest of paper is as follows: First, an introduction to the formal concept analysis is given. Following this an example class hierarchy based on the concept lattice is generated, and then the application of formal concept analysis to construction and reorganization of class hierarchy is discussed.

## 2 Formal Concept Analysis

Formal Concept Analysis is a field of applied mathematics based on mathematization of concept and conceptual hierarchy. The theory of Formal Concept Analysis was firstly introduced by Rudolf Wille, a German mathematician in 1982. Formal Concept Analysis (FCA), which also called concept lattice (or Galois lattice), is a kind of very effective tool for data analysis, and developed very rapidly in recent years. Until now, FCA have been applied to a number of different fields, such as medicine, biology and sociology[7].

### 2.1 Preliminaries

Formal concept analysis is based on the order theory, especially complete lattice theory, in mathematics. In order to thoroughly comprehend the theory and method of Formal Concept Analysis, some related theories and concepts should be explained firstly.

**Lattice theory:** Lattice theory is a branch of algebra, it is a kind of important tool to research algebra, geometry, topology, measure, functional, combinatorics, digital computer and fuzzy mathematics. Lattice theory is related to the following concepts.

**Ordered sets:** Assume  $P$  is a set, if the binary relation  $\leq$  of  $P$  satisfies the reflexivity, antisymmetry and transitivity, then it is called *order relation*(or shortly an *order*), the set of  $(P, \leq)$  which is composed of a series of order relation is called *ordered sets*.

**Table 1.** A representation of a formal context as a cross table.

	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$o_1$		$\times$				$\times$	$\times$
$o_2$			$\times$	$\times$	$\times$		
$o_3$			$\times$		$\times$	$\times$	$\times$
$o_4$	$\times$	$\times$					

**Lattice:** In an ordered set of  $(P, \leq)$ , if random two different elements  $x, y$  have *supremum*  $x \vee y$  and *infimum*  $x \wedge y$ , then the ordered set of  $(P, \leq)$  is called a *lattice*.

**Cover:** Assume  $a, b$  are random two different elements in a ordered sets of  $(P, \leq)$ , if  $a < b$ , but no  $x \in P$  satisfies  $a < x < b$  ( $x \neq a, x \neq b$ ), then call  $a$  is a *lower neighbor* of  $b$ , and  $b$  is a *above neighbor* of  $a$ , or call “ $b$  cover  $a$ ”, and record:  $a \prec b$ .

**Hasse diagram(line diagram):** In a limited ordered sets  $(P, \leq)$ , order relation  $\leq$  was completely decided by the element pairs which have cover relation(for example,  $a \prec b$ ). If the every element in set  $P$  is denoted by a circle in a plane. If and only if  $b$  cover  $a$ , then draw  $b$  above  $a$  ,and connect  $a, b$  through a line, then the graph is called a **line diagram** of ordered set of  $(P, \leq)$ , also called **Hasse diagram**. Through Hasse diagram, the relationship among the elements in ordered sets can be denoted.

## 2.2 Formal Concept and Galois Lattices

The basic structure of formal concept analysis is the context. A context is comprised of a set of objects, a set of attributes and a relation describing which objects possess which attributes. In the formal definition, the set of objects is denoted by  $\mathcal{O}$ , and the set of attributes is denoted by  $\mathcal{A}$ . To introduce the method FCA, we first have to define the term *context* or *formal context*.

**Definition 1** A *formal context* is a triple  $(\mathcal{O}, \mathcal{A}, \mathcal{R})$  that consists of two sets  $\mathcal{O}$  and  $\mathcal{A}$  and a relation  $\mathcal{R}$  between  $\mathcal{O}$  and  $\mathcal{A}$ . The element of  $\mathcal{O}$  are called the **objects** and elements of  $\mathcal{A}$  are called the **attributes** of the context. In order to express that an object  $o$  is in a relation with an attribute  $a$ , we write  $o\mathcal{R}a$  or  $(o, a) \in \mathcal{R}$  and read it as “the object  $o$  has the attribute  $a$ ”.

Contexts can be represented as cross tables, whose rows are headed by the objects, whose columns are headed by the attributes and whose cells are marked iff the incidence relation holds for the corresponding pair of object and attribute.

For instance, consider a context where this context is based on the set of objects  $\mathcal{O}$  and the set of their attributes  $\mathcal{A}$  as follows:

$$\begin{aligned}\mathcal{O} &= \{o_1, o_2, o_3, o_4\} \\ \mathcal{A} &= \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}\end{aligned}$$

and the incidence relation  $\mathcal{R}$  is given by the cross table(Table 1). The table should be read in the following way: Each  $\times$  marks a pair being an element of the incidence relation

$\mathcal{R}$ , e.g.  $(o_1, a_2)$  is marked because the object  $o_1$  has  $a_2$  as its attribute, whereas  $(o_1, a_3)$  is not marked because the object  $o_1$  does not have attribute  $a_3$ .

From the context a set of concepts is derived. Each concept is a pair  $(O, A)$  where  $O$  is a set of objects and  $A$  is a set of attributes. Both  $O$  and  $A$  are maximal sets that conform to each other. The central notion of FCA is the **formal concept**. A concept  $(O, A)$  is defined as a pair of objects  $O \subseteq \mathcal{O}$  and attributes  $A \subseteq \mathcal{A}$  which fulfill certain conditions. To define the necessary and sufficient conditions for a formal context, we define two derivation functions as follows:

**Definition 2** Let  $(\mathcal{O}, \mathcal{A}, \mathcal{R})$  be a context,  $O \subseteq \mathcal{O}$  and  $A \subseteq \mathcal{A}$ . The function intent maps a set of objects into the set of attributes common to the objects in  $O$  ( $\text{intent} : 2^{\mathcal{O}} \rightarrow 2^{\mathcal{A}}$ ), whereas extent is the dual for attributes sets ( $\text{extent} : 2^{\mathcal{A}} \rightarrow 2^{\mathcal{O}}$ ):

$$\begin{aligned}\text{intent}(O) &\stackrel{\Delta}{=} \{a \in \mathcal{A} \mid \forall o \in O : o \mathcal{R} a\}, \\ \text{extent}(A) &\stackrel{\Delta}{=} \{o \in \mathcal{O} \mid \forall a \in A : o \mathcal{R} a\}.\end{aligned}$$

Note that if  $O = \{\emptyset\}$  then  $\text{intent}(O) = O$ . Similarly, if  $A = \{\emptyset\}$  then  $\text{extent}(A) = A$ .

For  $O \subseteq \mathcal{O}$ ,  $\text{intent}(O)$  is the set of attributes owned by all classes of  $O$ . With  $O = \{o_1, o_3\}$ ,  $\text{intent}(O)$  is the set of attributes shared by both  $o_1$  and  $o_3$ , and more exactly  $\{a_6, a_7\}$ . Symmetrically, for  $A \subseteq \mathcal{A}$ ,  $\text{extent}(A)$  is the set of objects that own  $a_3$  and  $a_5$ , i.e.,  $\text{extent}(A) = \{o_2, o_3\}$ .

Objects from a context share a set of common attributes and vice versa. Concepts are pairs of objects and attributes which are synonymous and thus characterize each other. Concepts can be imagined as maximal rectangles(modulo permutation of rows and columns) in the context table. When looking at the cross table this property can be seen if rectangles totally covered with crosses can be identified, e.g. the four cells associated with  $o_1, o_3, a_6, a_7$  constitute such a rectangle. If we ignore the sequence of the rows and columns we can identify even more concepts.

**Definition 3** Let  $\mathcal{C} = (\mathcal{O}, \mathcal{A}, \mathcal{R})$  be a context. A pair  $c = (O, A)$  is called a concept of  $\mathcal{C}$  iff

$$(O \subseteq \mathcal{O} \wedge A \subseteq \mathcal{A}) \wedge (A = \text{intent}(O)) \wedge (O = \text{extent}(A)).$$

In other words a concept  $(O, A)$  is a pair where  $O \subseteq \mathcal{O}$ ,  $A \subseteq \mathcal{A}$ ,  $O$  is the extent of  $A$  and  $A$  is the intent of  $O$ . The set of all concepts of the context  $\mathcal{C} = (\mathcal{O}, \mathcal{A}, \mathcal{R})$  is denoted by  $B(\mathcal{C})$  or  $B(\mathcal{O}, \mathcal{A}, \mathcal{R})$ , i.e.,

$$B(\mathcal{C}) = \{(O, A) \in 2^{\mathcal{O}} \times 2^{\mathcal{A}} \mid \text{intent}(O) = A \wedge \text{extent}(A) = O\}.$$

Concepts are partially ordered by inclusion of extents(and intents) such that a concept's extent includes the extent of all its subconcepts(and its intent includes the intent of all of its superconcepts). That is, the set of formal concepts is organized the partial ordering relation  $\leq$  - to be read as “is a subconcept”- as follows:

**Table 2.** Concepts with its extensions and intensions from table 1.

Concept	Extension	Intension
$C_1$	$O$	$\emptyset$
$C_2$	$\{o_2, o_3, o_4\}$	$\{a_3\}$
$C_3$	$\{o_1, o_3\}$	$\{a_6, a_7\}$
$C_4$	$\{o_2, o_3\}$	$\{a_3, a_5\}$
$C_5$	$\{o_1\}$	$\{a_2, a_6, a_7\}$
$C_6$	$\{o_3\}$	$\{a_3, a_5, a_6, a_7\}$
$C_7$	$\{o_2\}$	$\{a_3, a_4, a_5\}$
$C_8$	$\{o_4\}$	$\{a_1, a_3\}$
$C_9$	$\emptyset$	$A$

**Definition 4** Let  $\mathcal{C} = (\mathcal{O}, \mathcal{A}, \mathcal{R})$  be a context. Given two concepts  $c_1 = (O_1, A_1)$ ,  $c_2 = (O_2, A_2) \in B(\mathcal{C})$ , the **subconcept/superconcept** relation  $\leq$  is defined by

$$(O_1, A_1) \leq (O_2, A_2) \Leftrightarrow O_1 \subseteq O_2 (\Leftrightarrow A_1 \supseteq A_2).$$

This relationship shows that the dualism exists between attributes and objects of concepts. A concept  $c_1 = (O_1, A_1)$  is a subconcept of concept  $c_2 = (O_2, A_2)$  iff the set of its objects is a subset of the objects of  $c_2$ . Or an equivalent expression is iff the set of its attributes is a superset of the attributes of  $c_2$ . This definition corresponds to the philosophical convention that a concept always has a smaller extension and a larger intension than any of its superconcepts. That is, a sub-concept contains fewer objects and more attributes than its super-concept.

The subconcept/superconcept relation “ $\leq$ ” on the set  $B(\mathcal{C})$  of all concepts of a context  $(\mathcal{O}, \mathcal{A}, \mathcal{R})$  satisfies the following three conditions<sup>1</sup>: *reflexivity*, *transitivity*, and *antisymmetry*.

### 1. reflexive

$$\forall c \in B(\mathcal{C})[c \leq c],$$

### 2. transitive

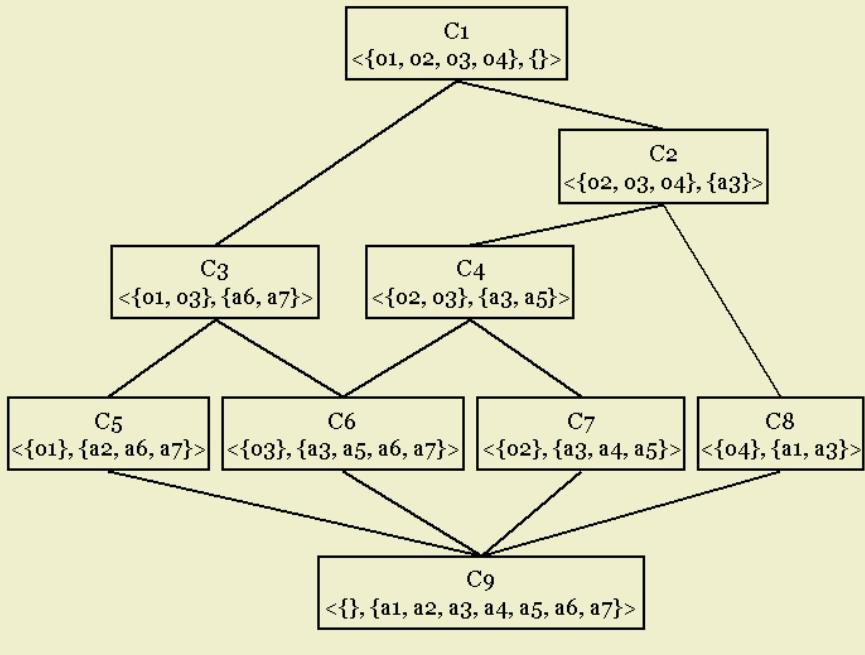
$$\forall c_1, c_2, c_3 \in B(\mathcal{C})[(c_1 \leq c_2) \wedge (c_2 \leq c_3) \Rightarrow (c_1 \leq c_3)],$$

### 3. antisymmetric

$$\forall c_1, c_2 \in B(\mathcal{C})[(c_1 \leq c_2) \wedge (c_2 \leq c_1) \Rightarrow (c_1 = c_2)].$$

These definitions now lead us to the following basic theorem of formal concept analysis which says that the set of concepts and the subconcept/superconcept relation defined above forms a complete lattice.

<sup>1</sup> Formal concepts associated to each other with the subconcept/superconcept relation form a class hierarchy, called *Concept Lattice*. In other words, a Concept Lattice is the ordered set of all formal concepts of a formal context.



**Fig. 1.** Example line diagram for the context of Table 1.

**Theorem 1** ([7]) Let  $\mathcal{C} = (\mathcal{O}, \mathcal{A}, \mathcal{R})$  be a context. Then  $(B(\mathcal{C}), \leq)$  is a complete lattice, the concept lattice of  $\mathcal{C}$ . Its infimum and supremum(also known as the meet and join, respectively) operation(for any set  $I \subset B(\mathcal{C})$  of concepts) are given by

$$\bigwedge_{i \in I} (O_i, A_i) = \left( \bigcap_{i \in I} O_i, \text{intent}(\text{extent}(\bigcup_{i \in I} A_i)) \right),$$

$$\bigvee_{i \in I} (O_i, A_i) = \left( \text{extent}(\text{intent}(\bigcup_{i \in I} O_i)), \bigcap_{i \in I} A_i \right).$$

This theorem says that the set of concepts produced by a context is a lattice and the meet and join of a group of concepts can be calculated by the above formula involving the intersection of attributes and objects.

Lattice operators *meet* and *join* provide the greatest lower bound(GLB) and the least upper bound(LUB) in the concept lattice respectively. Given two concepts  $(O_1, A_1)$  and  $(O_2, A_2)$  in the concept lattice, their *infimum* or *meet* is defined as:

$$(O_1, A_1) \wedge (O_2, A_2) = (O_1 \cap O_2, \text{intent}(\text{extent}(A_1 \cup A_2))),$$

$$= (O_1 \cap O_2, \text{intent}(O_1 \cap O_2)),$$

and their *supremum* or *join* as

$$(O_1, A_1) \vee (O_2, A_2) = (\text{extent}(\text{intent}(O_1 \cup O_2)), A_1 \cap A_2),$$

$$= (\text{extent}(A_1 \cap A_2), A_1 \cap A_2).$$

The results of  $c_3 \wedge c_4$  and  $c_4 \vee c_8$  from the context of Table 1 are:

$$\begin{aligned} c_3 \wedge c_4 &= (\{o_1, o_3\}, \{a_6, a_7\}) \wedge (\{o_2, o_3\}, \{a_3, a_5\}) \\ &= (\{o_3\}, \text{intent}(\{o_3\})) \\ &= (\{o_3\}, \{a_3, a_5, a_6, a_7\}) \\ &= c_6 \end{aligned}$$

$$\begin{aligned} c_4 \vee c_8 &= (\{o_2, o_3\}, \{a_3, a_5\}) \vee (\{o_4\}, \{a_1, a_3\}) \\ &= (\text{extent}(\{a_3\}), \{a_3\}) \\ &= (\{o_2, o_3, o_4\}, \{a_3\}) \\ &= c_2. \end{aligned}$$

Frequently, formal concepts of special interest are those concepts generated by a single attribute or by a single object from the context. These concepts are called *attribute concepts* and *object concepts* respectively<sup>2</sup>; they are useful because an object concept represents the smallest concept with this object in its extension while an attribute concept represent the largest concept with this attribute in its intension. As result, we have a lattice denoted by:

$$\mathcal{L} := (B(\mathcal{C}), \leq, \wedge, \vee, (\text{extent}(\mathcal{A}), \mathcal{A}), (\mathcal{O}, \text{intent}(\mathcal{O}))).$$

The lattice  $\mathcal{L}$  is called *Galois lattice* or *Formal concept lattice*.

### 2.3 Graphical Representation

A concept lattice (a concept hierarchy) can be represented graphically using line (or Hasse) diagrams. These structures are composed of nodes and links. Each node represents a concept with its associated intensional description. The links connecting nodes represent the subconcept/superconcept relation among them. This relation indicates that the parent's extension is a superset of each child's intension. A node covers all of the instances covered by the union of its descendants, making the concept hierarchy a subgraph of the partial ordering by generality. More abstract or general nodes occur higher in the hierarchy, whereas more specific ones occur at lower levels. A line diagram contains all the context information; an object has an attribute if and only if there is an upwards leading path from its object concept to this attribute concept. Figure 1 shows the line diagram for the context of Table 1.

The graph consists of nodes that represent the concepts and edges connections these nodes. Two nodes  $c_1$  and  $c_2$  are connected iff  $c_1 \leq c_2$  and there is no concept  $c_3$  with  $c_1 \leq c_3 \leq c_2$ . Although the concept lattice is a directed acyclic graph, the edges are not

---

<sup>2</sup> Looking at the definition of a formal concept one can easily see that for all  $O \subseteq \mathcal{O}$  the pair  $(\text{extent}(\text{intent}(O)), \text{intent}(O))$ , called an *object concept*, is a formal concept. The dual holds for all  $A \subseteq \mathcal{A}$ , i.e.  $(\text{extent}(A), \text{intent}(\text{extent}(A)))$ , called an *attribute concept*, is always a formal concept, too. Moreover, it is always true that  $\text{extent}(\text{intent}(O)) = O$  and  $\text{intent}(\text{extent}(A)) = A$ .

provided with arrowheads, instead the convention holds that the superconcept always appears above of all its subconcepts. For example, the line diagram shows that the node  $c_5$  and the node  $c_6$  are both subconcepts of the node  $c_3$ . Attributes and objects propagate along the edges, as a kind of inheritance. Attributes propagate along the edges to the bottom of the diagram and dually objects propagate to the top of the diagram. Thus the top element of a line diagram(the supremum of the context) is actually marked by  $(\mathcal{O}, \emptyset)$  if  $\mathcal{O}$  is the set of objects. The bottom element(the context's infimum) is marked by  $(\emptyset, \mathcal{A})$  if  $\mathcal{A}$  is the set of attributes.

## 2.4 Algorithms to Build the Concept Lattice

We present two main algorithms: a simple bottom-up one and a top-down algorithm to build a concept lattice.

**Bottom-Up Algorithm.** Given a set of objects  $A$ , the smallest concept with extent containing  $A$  is  $(\text{extent}(\text{intent}(A)), \text{intent}(A))$ . Thus, the bottom of the concept lattice is  $(\text{extent}(\text{intent}(\emptyset)), \text{intent}(\emptyset))$ - the concept consisting of all elements(often the empty set) that have all the attributes in the context relation.

The initial step of the algorithm is to compute the bottom of the concept lattice. The next step is to compute *atomic* concepts- smallest concepts with extent containing each of the objects treated as a singleton set. It is obvious that several calculations result in the same set of objects, meaning that these objects form the extent of the same concept.

The algorithm then closes the set of atomic concepts under join: Initially, a work-list is formed containing all pairs of atomic concepts  $(c', c)$  such that  $(c \not\leq c') \wedge (c' \not\leq c)$ . While the work-list is not empty, remove an element of the work-list  $(c_0, c_1)$  and compute  $c'' = c_0 \vee c_1$ . If  $c''$  is a concept that is yet to be discovered then add all pairs of concepts  $(c'', c)$  such that  $(c \not\leq c'') \vee (c'' \not\leq c)$  to the work-list. The process is repeated until the work-list is repeated.

```

Algorithm BottomUpAlgorithmToBuildConceptLattice
Begin
  edges  $\leftarrow \emptyset$ ;  $S \leftarrow B(\mathcal{O}, \mathcal{A}, \mathcal{R})$ ;
  for all  $c_1 \in S$  do
    for all  $c_2 \in S - \{c_1\}$  do
      if  $(c_1 \leq c_2) \wedge (\nexists c_3 \in S - \{c_1, c_2\}[(c_1 \leq c_3) \wedge (c_3 \leq c_2)])$  then
        edges  $\leftarrow$  edges  $\cup \{(c_1, c_2)\}$ ;
      end_if
    end_for
  end_for
End
```

**Top-Down Algorithm.** When constructing the lattice  $\mathcal{L}$ , we proceeds by a top-down specialization, beginning with the most general node. For each node, the algorithm computes all its sub-nodes. The algorithm scheme is the following( $F$  denotes the FIFO list):

**Algorithm TopDownAlgorithmToBuildConceptLattice****Begin**
 $F \leftarrow (\mathcal{O}, \emptyset)$ 
**Repeat**Remove  $(O_k, A_k)$  in  $F$  and Insert it in  $\mathcal{L}$ Compute  $C_k$  the set of sub-node of  $(O_k, A_k)$ Insert each new node created in  $F$ **Until**  $F$  is empty**End**

The proposed algorithm constructs the lattice step by step, beginning with the most general concept. A node(concept) is computed by each of its predecessor. Consider  $(O_k, A_k)$  a node in the lattice and  $R_k = \mathcal{R} \setminus_{O_k \times (\mathcal{A} - A_k)}$ , the restriction of the binary relation  $\mathcal{R}$  to the two subsets  $O_k$  and  $\mathcal{A} - A_k$ . Consider  $Col_k \subseteq \mathcal{A} - A_k$ , the set of labels of a maximal subset in  $R_k$ , then the node  $(O_{k_i}, A_{k_i})$  is created as a sub-node of  $(O_k, A_k)$  as follows:

$$\begin{aligned} O_{k_i} &= O_k - \{o \in O_k \mid \exists a \in Col_k : (o, a) \notin R_k\}, \\ A_{k_i} &= A_k \cup Col_k. \end{aligned}$$

If  $m = |\mathcal{O}|$  and  $n = |\mathcal{A}|$  then at most  $2^{\min(m,n)}$  concepts have to be generated which results in an unacceptable complexity.

### 3 Considering Concept Lattice AS Class Hierarchy

A concept lattice can be considered as a class hierarchy as follows<sup>3</sup>:

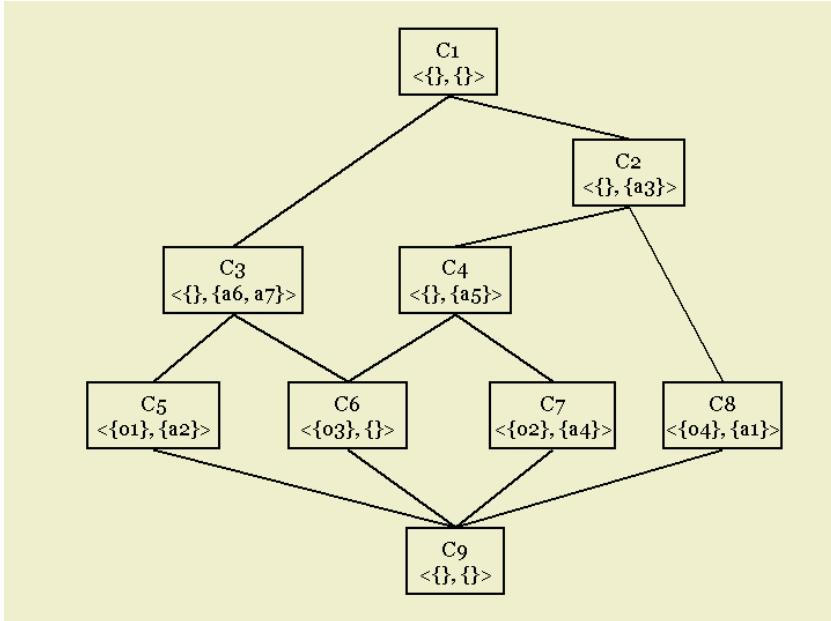
- a concept  $c = (O, A) \in B(\mathcal{C})$  is a class with extension  $O$  and intension  $A$ .
  - $\leq$  is the specialization relationship between two classes;
  - $\wedge$  is the specialization operation(intersection of extensions) of two classes;
  - $\vee$  is the generalization operation(intersection of the attribute sets) of two classes;
  - $(\text{extent}(\mathcal{A}), \mathcal{A})$  is the bottommost class of the hierarchy.
- Intension is the union of all attributes; extension may be empty, i.e.,  $(\emptyset, \mathcal{A})$ ;
- $(\mathcal{O}, \text{intent}(\mathcal{O}))$  is the topmost class of the hierarchy.

Extension is the union of all base extensions; intension may be empty, i.e.,  $(\mathcal{O}, \emptyset)$ .

However, the concept lattice itself is not adequate for class hierarchies because of the redundancy in the representation. For a pair  $c = (A, B)$ , the elements of  $A$  will be present in every ancestor of  $c$  and symmetrically, the elements of  $B$  will appear in every descendant. For the class hierarchy design problem, the duplication should therefore be eliminated and be computed when needed by the inheritance mechanism.

---

<sup>3</sup> In the case of object-oriented class hierarchies, concepts correspond to classes and their properties which can be either attributes or methods. All properties have a name and other characteristics such as *signature*, and in the case of methods they may have a *body* or *code*(a set of instructions). The signature of an attribute is its data type. The signature of a method is the ordered list of its parameter types and possibly its return type.



**Fig. 2.** An Example of Inheritance Concept Lattice.

For a concept  $c = (A, B)$ , let  $N(A)$  be the non-redundant elements in  $A$ , and  $N(B)$  the non-redundant elements in  $B$ . A class  $g$  will appear in  $N(A)$  if the corresponding concept  $c$  is the greatest lower bound of all concepts containing  $g$ . Symmetrically, a property  $m$  will appear in  $N(B)$  if the corresponding concept  $c$  is the least upper bound of all concepts containing  $m$ . The lattice property guarantees the existence of the greatest lower bound and the least upper bound.

**Definition 5** A inheritance concept lattice ICL is isomorphic to a concept lattice  $\mathcal{L}$ . For a concept  $c = (X, Y) \in \mathcal{L}$ , let  $N(X)$  be the non-redundant elements in  $X$ , and  $N(Y)$  the non-redundant elements in  $Y$ :

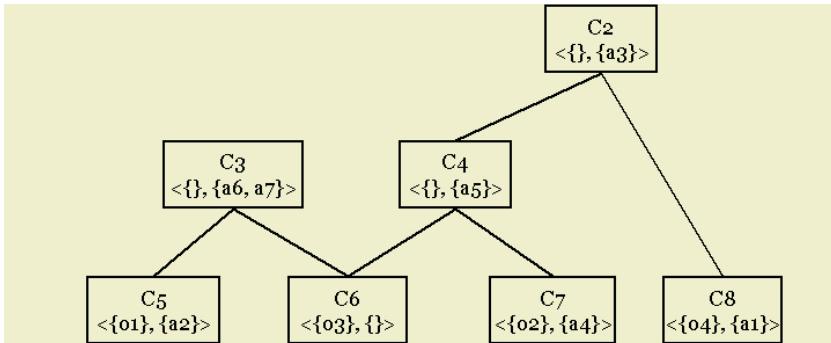
$$\begin{aligned} N(X) &= \{x \in \mathcal{O} \mid \text{intent}(\{x\}) = Y\} \\ N(Y) &= \{y \in \mathcal{A} \mid \text{extent}(\{y\}) = X\} \end{aligned}$$

In other words,

$$\begin{aligned} N(X) &= X - \bigcup_{c' \in \text{Subclasses}(c)} X' \\ N(Y) &= Y - \bigcup_{c' \in \text{Superclasses}(c)} Y' \end{aligned}$$

where we note  $c' = (X', Y')$ . A inheritance concept lattice is defined as the set of concepts  $(N(X), N(Y))$ .

The inheritance concept lattice(ICL) is obtained from the concept lattice  $\mathcal{L}$  by replacing each concept  $(A, B)$  with the pair  $(N(A), N(B))$ . We call this lattice an inheritance concept lattice.



**Fig. 3.** An Example of Galois Sub-Hierarchy.

Figure 2 is the ICL for the example of figure 1. The Concept Lattice and Inheritance Concept Lattice are isomorphic lattices. From a class design perspective, the idea is to consider each concept of the generated ICL as a class and the edges of the Hasse diagram as the class-subclass relations. The ICL also shows where each property should be declared. One important property of the ICL is that each property appears exactly in one place. We obtain in such a manner a maximal factorization of the common properties of the classes.

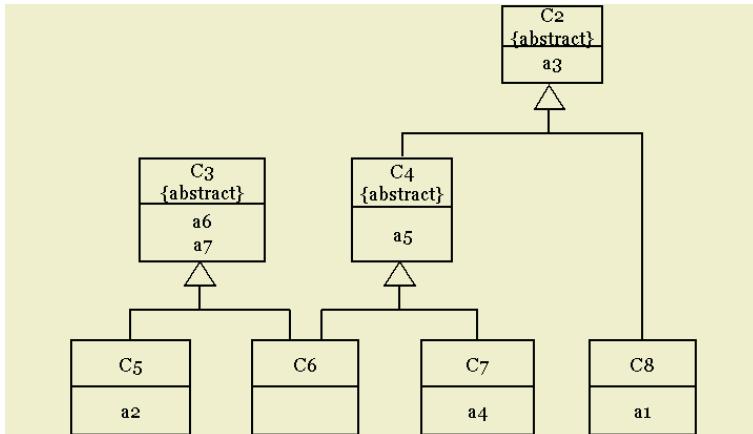
However, some of the generalizations of the concept lattice are empty in that they do not possess their own attributes or objects: all their attributes appear in at least one super-concept(inheritance) and dually, all their objects appear in a sub-concept(extension inclusion). These concepts could be eliminated without loss of information thus leading a structure called a *Galois Sub-Hierarchy* which corresponds to the union of the sets of attribute concepts and object concepts of the galois lattice.

**Definition 6** *The Galois Sub-Hierarchy GSH( $\mathcal{R}$ ) is the order deduced from Inheritance Concept Lattice ICL by removing “empty” vertices , i.e. vertices such that  $(\emptyset, \emptyset)$ .*

The Galois Sub-Hierarchy(GSH) can be generated from the concept lattice by eliminating the pairs which have empty  $N(A)$  and  $N(B)$  sets. The line diagram of the galois sub-hierarchy is depicted in Figure 3. Although the maximal factorization is preserved in the galois sub-hierarchy, the lattice property is not always preserved. That is, this structure is not necessary a lattice but, when interpreting its nodes as classes, it is maximally factored, consistent with specialization. It could also be considered as another kind of normalized class hierarchy form, as shown in figure 4.

## 4 Reorganization of Class Hierarchy

Inheritance may be defined as the mechanism used in object-oriented languages to implement specialization/generalization relationships between classes. In particular, it offers a way to share(factorize) properties(methods and attributes) among classes. A class hierarchy based on the inheritance relationship usually constitutes the backbone of an



**Fig. 4.** Class Hierarchy organized from the galois sub-hierarchy(fig.3).

object-oriented system. Though the initial version of a class hierarchy is often built with care, it may contain some flaws regarding the factorization of properties. Furthermore, as the hierarchy is modified through corrective or evolutive maintenance operations, more flaws may be introduced that may reduce both system performance and ease of maintenance. A number of different operations relevant to the reconstruction or reorganization of class hierarchies(Galois sub-hierarchy) is possible. Possible transformations are the following:

1. Adding and Deleting classes which do not have extensions but intensions of their own.

An inheritance hierarchy may contain classes which does not have extensions of their own(i.e., *abstract classes*). In other words, for these classes the extension is given by the union of their subclass extensions. These classes can be added or deleted without loss of information. When the *abstract classes* are deleted, their attributes must be moved to their direct subclasses.

2. Vertically(Horizontally) merging and splitting of classes.

Vertically merging means merging a class with its direct superclass. The vertically merged class is formed by the union of the extensions and of the intensions. Classes can also be merged horizontally if they have a direct specialization relation to the same superclass. The horizontally merged class is formed by the union of the subclass extensions and of the intensions.

## 5 Conclusions and Future Works

A class hierarchy based on the inheritance relationship usually constitutes the backbone of an object-oriented system. Any kind of automated help in building,reorganizing or maintaining “good” class hierarchies can thus be of interest and can have applications in several important research areas of object-oriented technology.

In this article, we have presented our research-in-progress concerning the normalized form of object-oriented class hierarchies which is helpful to build and reorganize class hierarchies. The whole of this work is based on the concept lattice of the Formal Concept Analysis(FCA) which allows to construct “well defined” class hierarchies with maximally factorized properties. The use of FCA in the design of class hierarchies provides many benefits to the object-oriented software designer. Each class constructed by FCA is necessary based on the property cross table and so only essential complexity is contained in the final class structure. Each class contains only those factors necessary for its identity. This minimal approach to class construction provides classes that are functionally encapsulated, that is cohesive.

FCA has typically been applied in the field of software engineering. In [17] the authors present a broad overview by describing and classifying academic papers that report the application of FCA to software engineering: They classified 42 publications using FCA. The found categories are split in the *Early phase activities*(Requirement analysis, Component retrieval software, Formal specification, and Visualizing Z specification via FCA) and the *Software maintenance*(Dynamic analysis, Application to legacy systems, Reengineering class hierarchies, and Conceptual analysis of software structure).

The goal of our work is to design and implement a high level environment for building and maintaining(reorganizing, merging, etc.) class hierarchies having formal properties ensured by the Galois sub-hierarchy that underlies the structure used to constrain them. Our current work consists in studying for transformations used to build and reorganize well structured class hierarchies from any kind of existing ones, and allowing programmers or designers to perform basic as well as complex manipulations on their hierarchies in a safe way, with the insurance that they will not damage the original class hierarchies.

## References

1. Casais, E., 'Managing class evolution in object-oriented systems', In O.Nierstrasz and D.Tsichritzis, editors, Object-Oriented Software Composition, pp.201-244, Prentice Hall, 1995.
2. Jonson, R.E. and Opdyke, W.F. "Refactoring and Aggregation", ISOTAS'93 Proceedings, 1993.
3. Lance Tokuda and Don Batory, "Evolving object-oriented designs with Refactorings," Journal of Automated Software Engineering, Vol.8, No.1, pp.89-120, 2001.
4. Paul L. Bergstein, "Object preserving class transformation", SIGPLAN Notices, Vol.26, No.11, 1991.
5. J.-B. Chen and S. Lee, "Generation and reorganization of subtype hierarchies," Journal of Object-Oriented Programming, 8(8), 1996.
6. H. Dicky, C. Dony, M. Huchard, and T. Libourel, "Ares, Adding a class and restructuring inheritance hierarchies," Technical Report, LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5, France, February 1995.
7. B. Ganter and R. Wille, "Formal Concept Analysis, Mathematical Foundations," Springer-Verlag, 1999.
8. R. Wille, "Concept lattices and conceptual knowledge systems," Computers Math. Applic, 23, 493-513, 1992.
9. B. Birkhoff, Lattice Theory, American Mathematical Society Colloquium Publ., Vol.25, 1973.

10. Spangenbergs, N., & K.E. Wolff. Concept lattices as indicators of change in the therapeutic process: does formal concept analysis of repertory grids represent a paradigm change of data evaluation? in Psychoanalytic research by means of formal concept analysis, N. Spangenbergs, K.E. Wolff (eds.), Sigmund-Freud- Instituts, Lit Verlag, Munster, 1999.
11. Ganter B. & R. Wille. Conceptual Scaling, in Applications of Combinatorics and Graph Theory in the Biological and Social Sciences, F. Roberts (ed.) Springer, New York, 1989.
12. Kent R.E. & C. Neuss. Creating a 3D Web Analysis and Visualization Environment, Computer Networks and ISDN Systems, Vol. 28: 109-117, 1995.
13. Faid M., R. Missaoui & R. Godin. Mining Complex Structures Using Context Concatenation in Formal Concept Analysis. Second International KRUSE Symposium (KRUSE'97), Vancouver, British Columbia, August 11-13, 1997.
14. Schmitt, I., & G. Saake. Merging Inheritance Hierarchies for Schema Integration based on Concept Lattices, Technical Report, Faculty of Information, University of Magdeburg, 1997.
15. Deogun J.S., V.V. Raghavan, & H. Sever. Association Queries and Formal Concept Analysis, The Sixth International Workshop on Rough Sets, Data Mining and Granular Computing (in conjunction with JCIS'98), 23-28 October, Research Triangle Park, NC, USA, 1998.
16. Priss U. Efficient Implementation of Semantic Relations in Lexical Databases, Computational Intelligence, Vol. 15-1, 1999.
17. T. Tilley, R. Cole, P. Becker, and P. Eklund. "A Survey of Formal Concept Analysis Support for Software Engineering Activities," In G. Stumme, editor, Proceedings of the First International Conference on Formal Concept Analysis-ICFCA'03, Springer-Verlag, February 2003.

# **QoS-Guaranteed DiffServ-Aware-MPLS Traffic Engineering with Controlled Bandwidth Borrowing**

Youngtak Kim and Chul Kim

Dept. of Information and Communication Engineering

Graduate School, Yeungnam University

214-1, Dae-Dong, Kyungsan-Si, Kyungbook, 721-749, South Korea

ytkim@yu.ac.kr, goldfe@ownnuri.net

**Abstract.** In this paper we propose an integrated traffic engineering mechanism based on the DiffServ-aware-MPLS for Next Generation Internet that provides guaranteed bandwidth with controlled bandwidth borrowing among LSPs. In the proposed scheme, each DiffServ class-type is managed with per-class-type queuing controlled by committed data rate (CDR), peak data rate (PDR) and related burst sizes. A hierarchical DiffServ packet scheduler controls the aggregated packet flow of multiple DiffServ class types, where a priority-based scheduler is used for overall traffic, and a weight-based scheduler is used for the AF traffic flows. Another packet scheduler for MPLS LSPs is also organized with hierarchical configuration of weight-based scheduling and priority-based scheduling. We also propose a controlled bandwidth borrowing mechanism with extended MPLS signaling for maximized resource utilization. From the simulation results, we could see the usefulness of the proposed scheme to guarantee QoS provisioning and maximize resource utilization.

## **1 Introduction**

### **1.1 Internet Traffic Engineering**

MPLS [1-3] provides various attractive features of traffic engineering based on connection-oriented explicitly labeled path. The traffic trunk of aggregated traffic flows of the same class type can be easily mapped onto LSPs, and a set of attributes can be associated with the traffic trunks that modulate their behavioral characteristics. Also, a set of attributes can be associated with the resources that constrain the placement of LSPs and traffic trunks across them. MPLS allows flexible traffic aggregation, and it is relatively easy to implement a constraint-based routing with lower overhead.

To provide the traffic engineering capability, the existing signaling and routing protocol modules must be expanded. As routing and signaling protocol with traffic engineering (TE) extensions, OSPF-TE [4], RSVP-TE [5], IS-IS-TE[6], CR-LDP[7] are under standardization in IETF. To support the inter-domain traffic engineering, the TE extensions to BGP-4 protocol also has been proposed [8]. The signaling and routing protocols with TE extensions basically provide mechanisms of maintaining the link state database according to the specified TE parameters, such as physical distance, available bandwidth, allocated bandwidth, residual error rate, resource color, and shared risk link group (SRLG) identifier.

## 1.2 Differentiated Service (DiffServ)

The DiffServ technology has been developed to provide differentiated quality-of-service (QoS) according to the user's requirements and necessity [8-11]. The main focus of the differentiated service provisioning is to protect the premium service traffic under network congestion by giving relatively higher forwarding priority than other usual best-effort traffic. For each differentiated service, Per-Hop-Behavior (PHB) is specified at each IP router, and differentiated configurations of packet queuing and scheduling are specified by multiple parameters, such as the source/destination address range, service type, protocol identifier, and port number range. 64 different classes with distinct DiffServ Code Points (DSCP) are defined with 6-bit field in the IP packet header.

In order to simplify the classification of DiffServ, a set of DiffServ classes is defined as a *class-type*, where the classes in the same class-type possess the common aggregate maximum and minimum bandwidth requirements to guarantee the required performance level. The DiffServ class-types that are proposed in IETF can be grouped into 8 major categories: 2 kinds of network control traffic (NCT0, NCT1), expedited forwarding (EF), 4 kinds of assured forwarding (AF1, AF2, AF3, and AF4)[12], and best-effort forwarding (BEF).

For the differentiated and classified processing of packets at each IP routers, the DiffServ-aware routers should process packets according to the DiffServ class-types, metering and coloring, class-based-queuing with algorithmic packet drop, packet scheduling, and optional traffic shaping [11]. There is no maximum or minimum bandwidth requirement to be enforced at the level of an individual class within a class-type.

## 1.3 DiffServ-Aware MPLS Traffic Engineering

The mapping of DiffServ class-types into MPLS LSP (Label Switched Path) can be implemented in either E-LSP (Exp-inferred-LSP) [10] or L-LSP (Label-only-inferred LSPs) model. In E-LSP model, multiple class-types (ordered aggregates) are mapped onto an MPLS LSP, and the EXP field of the MPLS Shim header conveys the PHB to be applied to the packet at each LSR; the PHB conveys both information of the packet scheduling treatment and the drop precedence. In L-LSP model, each LSP only transports a single class-type, so the packet treatment is inferred exclusively from the packet label value. The EXP field of the MPLS shim header specifies the packet drop precedence.

The E-LSP model has merits of easier connection management and protection; the creation of a single LSP for end-to-end services for a customer is easier than the setup, maintenance, administration, and monitoring of multiple LSPs for each class-type [6]. Also, E-LSP model requires reduced number of LSPs needed to deploy end-to-end services in an MPLS network. The path protection and switching mechanisms are more easily applied to a single LSP than a group of related LSPs. Finally, the bandwidth borrowing among the classes of a class-type from a customer is much easier.

The major problems to be solved in the DiffServ-aware-MPLS traffic engineering are (i) how to determine the operational parameters for the guaranteed QoS, and (ii)

how to configure the DiffServ-aware-MPLS LSRs along the path. In Section 2, we propose a method to determine the operational parameters.

### 1.4 Bandwidth Borrowing

In the circuit switched network, the under-utilization of fixed-bandwidth circuit has been the major problem. Since MPLS LSP also uses a connection-oriented virtual circuit, it also has the same potential problem, especially when the established LSP is kept with a strict bandwidth reservation regardless of the actual utilization. To solve this problem several schemes have been proposed such as bandwidth overbooking[13], bandwidth borrowing[14], and bandwidth sharing[15].

Bandwidth overbooking in MPLS network can be performed in off-line or on-line. This scheme requires the information of the traffic demand prediction, the real-time link state information and the link state routing. Since the bandwidth over-booking always has the risk of coincident peak bandwidth requests from multiple LSPs, the possible malicious user-behavior, and the inaccuracy of prediction of traffic demands, it is not a good approach of traffic engineering for the connection-oriented tunnel LSP that should provide guaranteed QoS. It may also suffer from the potential under-utilization. Bandwidth sharing (e.g., CBQ) [14,15] has considered about the distribution of resource to constituent flows. This scheme can be used in DiffServ-aware MPLS network in order to provide a high utilization of E-LSP with flexible bandwidth sharing among the constituent micro packet flows.

The major problems in the bandwidth borrowing are (i) how to collect the information of extra-available bandwidth along the LSP which is hierarchically stacked, and (ii) how to re-distribute the extra-available bandwidth to the constituent inner LSPs.

The rest of the paper is organized as follows. In section 2, the integrated traffic engineering scheme for DiffServ-aware-MPLS is proposed. In section 3, the simulation results are analyzed, and finally we conclude in section 4.

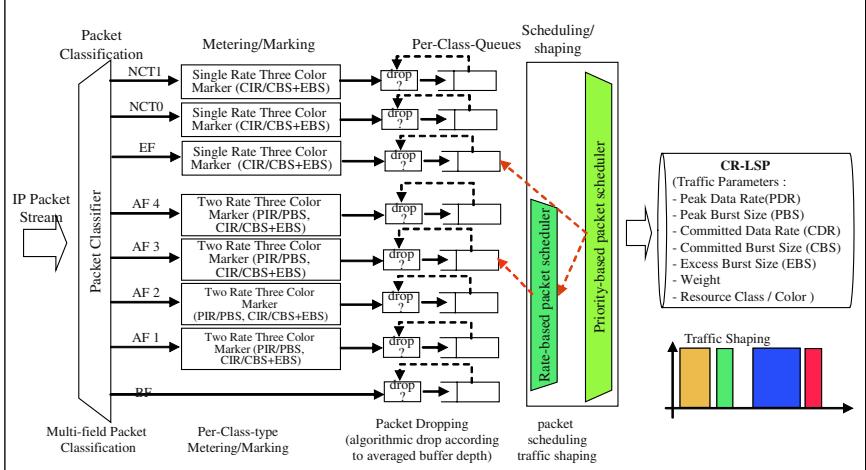
## 2 Integrated Traffic Engineering for DiffServ-Aware-MPLS

In this paper, we propose a traffic engineering mechanism for Next Generation Internet that achieves two major conflicting objectives: guaranteed QoS provisioning and maximized bandwidth utilization. The proposed traffic engineering is based on Diff-Serv-aware-MPLS, where each DiffServ class-type is managed with per-class-type policing (metering and marking) and queuing with committed data rate (CDR), peak data rate (PDR), and related buffer sizes. We also propose a controlled bandwidth borrowing scheme for improved utilization of hierarchical LSPs with periodic discovery of unused bandwidth of physical link or tunnel LSPs, and redistribution the extra available bandwidth among their constituent flows.

### 2.1 Overall Packet Processing Structure of DiffServ Packet Handling

Fig. 1 shows the per-class-type packet processing structure of DiffServ-aware MPLS routers. The major functional blocks of the DiffServ packet processing are packet classifier, meter/marker, per-class-type queue with separated buffer, DiffServ packet

scheduler and optional traffic shaper. For each DiffServ class-type, the individual packet processing profile is defined to specify the traffic parameters, the meter/marker type, queuing mechanism, buffer size, threshold values, priority, and weight. The traffic parameters of the overall aggregated traffic are given to the packet scheduler and traffic shaper that control the bandwidth of the aggregated packet flow.



**Fig. 1.** Packet Processing in DiffServ-aware IP Router.

We assume that the user defines the traffic/QoS parameters and the aggregation of multiple DiffServ class-types by the service level agreement (SLA). Also, the traffic parameters of each micro-flow are specified at the DiffServ configuration, and the traffic parameters and the priority of the overall aggregate packet flow are also defined. The packet classification is implemented with multi-field classifier that uses multiple fields in the IP packet header to determine the differentiated per-hop-behavior (PHB). The IP source address prefix (address and prefix length), destination address prefix, upper-layer protocol, TCP/UDP/SCTP source and destination port range, and ToS (Type of Service) or DSCP (DiffServ Code Point) fields are usually used in the packet classification.

## 2.2 Integrated Traffic Engineering for Guaranteed Bandwidth

In order to guarantee the bandwidth of each DiffServ class-type, we are using separated policing (metering & marking) and queuing modules for each DiffServ class-type. The packet flow of each class-type is specified with its traffic parameters that include committed data rate (CDR) / committed burst size (CBS) + excess burst size (EBS), and peak data rate (PDR) / peak burst size (PBS). Also the priority and the weight of each class-type are allocated for the packet scheduling of the aggregated packet flow.

The CDR of each DiffServ class-type is guaranteed by the specification of the aggregated CDR of traffic engineering to be greater than or at least equal to the summation of the CDR of packet flows of the class-type. The PDR of the aggregated packet

flow is specified by the user at the service-level specification (SLS) considering his own traffic engineering policy. The DiffServ-aware-MPLS TE will guarantee the CDR of the aggregated traffic flow with possible extra bandwidth up to the PDR. The amount of the extra bandwidth depends on the bandwidth utilization of other LSPs in the TE-LSP.

### 2.3 End-to-End Transfer Delay and Jitter

The end-to-end packet transfer delay and the jitter are important constraints of traffic engineering parameters which are specified at the service-level agreement. As an example, the target end-to-end transfer delay of VoIP is specified as 150 ms, while the jitter is specified as 60 ms. The specification of the end-to-end transfer delay and the jitter may be different according to the application service classes. In the case of best effort service, the timing constraints are not specified.

The *end-to-end transfer delay* is composed of the propagation delay that is determined by the physical distance, and the packet switching delays at each Diff-Serv/MPLS-LSR. The switching delay is composed of the processing time and queuing delay time at the DiffServ policing (metering and marking), DiffServ queuing, MPLS policing and queuing with packet scheduling. Also, the jitter is generated by the queuing delay variance at each class-type queue, and by the packet scheduling. Especially in the priority-based packet scheduling, the higher-priority traffic flow contributes the jitter and the queuing delay of the lower-priority packet flows.

In the DiffServ-aware-over-MPLS, the queuing delay is mostly generated at the ingress LER (label edge router) where randomly arrived packets are queued, and scheduled according to the bandwidth of the designated LSP. Since the packet flow is usually controlled by the traffic shaper at ingress LER, the queuing delay at intermediate LSR is not significant. With this consideration, we can distribute the end-to-end total switching & queuing delay to each LSR on the route, and use the switching & queuing delay to determine the maximum allowed queuing delay at each LSR. The buffer sizes of the DiffServ queue and the MPLS packet scheduling queue are determined by this maximum allowed queuing delay at each node. With the same analogy, we can distributed the target jitter value to each LSRs, and control the buffer usage at each LSR according to the allowed jitter amount at the packet switching node.

The queuing delay is mostly determined by the bandwidth utilization. Since the DiffServ packet input at the ingress LER has random distribution and the packet transmission from DiffServ-aware MPLS router has deterministic processing time that is specified by CDR, we can use M/D/1 queuing model in the analysis of the queuing delay at the ingress LER node. In M/D/1 queue model, the average waiting time is calculated by

$$t_w = \frac{\rho}{2(1-\rho)} \bullet s = \frac{\rho}{2(1-\rho)} \bullet \frac{B}{R} \quad (1)$$

where  $\rho$  is utilization of the connection,  $s$  is the service time of a packet that is calculated by the bandwidth or transmission rate of the connection,  $B$  is the packet size, and  $R$  is the transmission rate. The mean time in a node is simply equal to the sum of the mean service time and the waiting time, i.e.

$$t_q = t_w + s = \left( \frac{\rho}{2(1-\rho)} + 1 \right) \bullet s = \frac{(2-\rho)}{2(1-\rho)} \bullet \frac{B}{R} \quad (2)$$

Using this relationship, we can adjust the packet size of the information flow or control the utilization of connection to limit the average queuing delay at a certain limit. For example, in a VoIP packet stream with packet size 128 bytes/packet and 128Kbit/sec transmission rate, the packet service time becomes 24 ms at the ingress LER when the link utilization is 80%. The packet service time increases sharply as the link utilization increases; when the link utilization is 95%, the service time at ingress LER becomes 84 msec which may be beyond the limit to guarantee the end-to-end transfer delay within 150ms.

## 2.4 Packet Loss Ratio and Buffer Size

Packet loss occurs when the buffer is overflowed at the DiffServ class-type queue or at the MPLS packet scheduling queue. ITU-T Y.1540/Y.1541 recommendations specifies the target performance values of IP-based network QoS standards, where the IP loss rate is required less than  $10^{-3}$ , while the IP error rate is required less than  $10^{-4}$ .

In order to keep the packet loss ratio at a certain limit under the normal network operation status, the probability of the buffer overflow must be limited. In M/D/1 queue model, the probability of the heavy traffic buffer overflow is approximately defined by [18]

$$\Pr\{\text{System size} > x\} = Q(x) = \exp\left[-2x\left(\frac{1-\rho}{\rho}\right)\right] \quad (3)$$

where  $x$  defines the number of packets in the buffer and  $\rho$  is the utilization of the connection. In the same example of the previous VoIP packet flow (128 bytes packet, 128Kbps and utilization of 80%), the buffer size to limit the probability of overflow less than  $10^{-3}$  is determined by  $-2x\left(\frac{\rho}{1-\rho}\right) < \ln(10^{-3})$ , and  $x$  should be less than 20.7

packets. So, the buffer size should be larger than  $21 \times 128$  bytes = 2,688 bytes. The calculated buffer size must be configured as the allocated buffer size at the ingress MPLS LER.

## 2.5 Controlled Bandwidth Borrowing

In order to efficiently distribute the extra bandwidth to tunnel LSPs, we use a weight-based recursive distribution scheme. We assume that each inner LSP of a given tunnel LSP is assigned a weight value in accordance with the network management policy or Service Level Agreement (SLA).

Let us denote the extra bandwidth of a physical link /port as *extraAvailableBW* and the number of tunnel LSP as *k*. Then, the available bandwidth of each tunnel LSP is determined as follows:

$$availableBW\_LSP_i = \frac{w_i}{\sum_k w_k} \times extraAvailableBW \quad (4)$$

Each tunnel LSP has the extra bandwidth of  $availableBW\_LSP_i$ . If the tunnel LSP has  $m$  inner LSPs, the  $availableBW\_LSP_i$  is distributed recursively to each inner  $LSP_{ij}$  as follows:

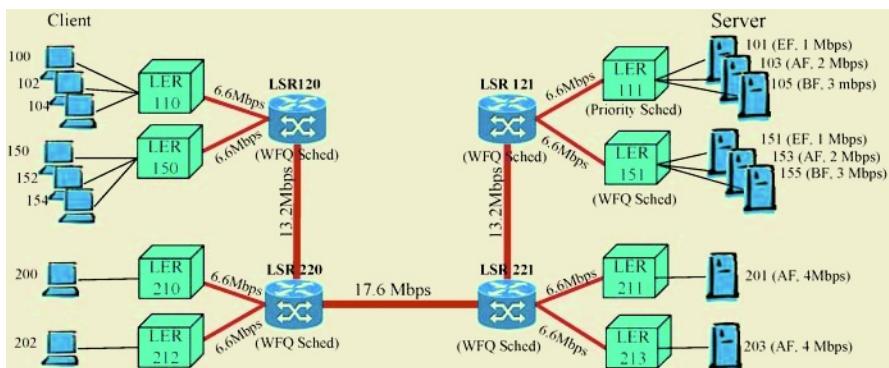
$$availableBW\_LSP_{ij} = \frac{w_{ij}}{\sum_m w_{im}} \times availableBW\_LSP_i \quad (5)$$

where  $w_i$  is the weight value of tunnel LSP  $i$  and weight value in the  $j^{\text{th}}$  internal LSP is  $w_{ij}$ . The network management system has to determine the weight value of each tunnel LSP.

### 3 Analysis of the Proposed Scheme

#### 3.1 Simulation Network Topology

DiffServ-aware-MPLS provides the capability of the micro-flow traffic engineering for each class-type in a aggregated packet stream with a LSP. Fig. 2 shows the simple network topology to test the functions of DiffServ-over-MPLS traffic engineering on network simulator, and Table 1 shows the traffic generation scenario between node pair. We used the NIST GMPLS Network Simulator, GLASS (GMPLS Lightwave Agile Switching Simulator) [16] to verify the correct operations of the proposed scheme.



**Fig. 2.** Simulation topology for DiffServ-over-MPLS.

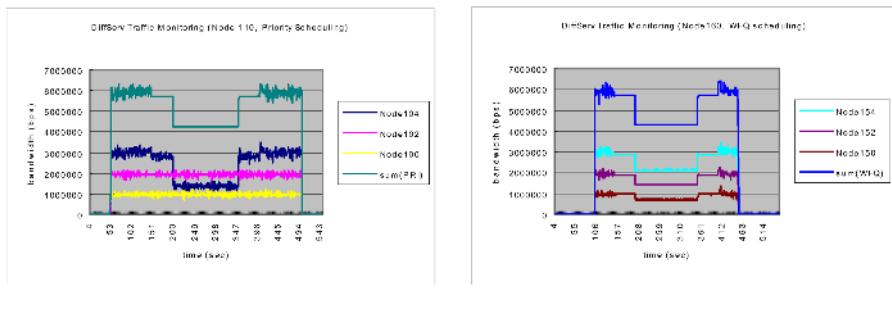
In order to test the guaranteed provisioning of bandwidth and QoS parameters, the simulation topology has bottleneck link between LSR 220 and LSR 221 through which all LSPs between LER pairs pass. The traffic generations also have different timing to simulate a gradually fluctuating network condition. During simulation time of 200 ~ 350 sec, the utilization of this link is increased up to 95%.

**Table 1.** Traffic Generation of DiffServ-over-MPLS.

Src - Dest	Class Type	Traffic Parameters (DiffServ) [Mbps]	Priority/Weight (DiffServ)	Packet scheduling (DiffServ)	Traffic Generation Period [sec]
100-101	EF	PDR: 1.5 CDR: 1.0	P: 5	Priority	50 ~ 500
102-103	AF	PDR: 3.0 CDR: 2.0	P: 3	Priority	50 ~ 500
104-105	BE	PDR:4.5 CDR:3.0	P: 1	Priority	50 ~ 500
150-151	EF	PDR: 1.5 CDR: 1.0	W: 1	WFQ	100 ~ 450
152-153	AF	PDR: 3.0 CDR: 2.0	W: 2	WFQ	100 ~ 450
154-155	BE	PDR:4.5 CDR:3.0	W: 3	WFQ	100 ~ 450
200-201	AF	PDR: 6.0 CDR: 4.0	W: 4	WFQ	150 ~ 400
202-203	AF	PDR: 6.0 CDR: 4.0	W: 4	WFQ	200 ~ 350

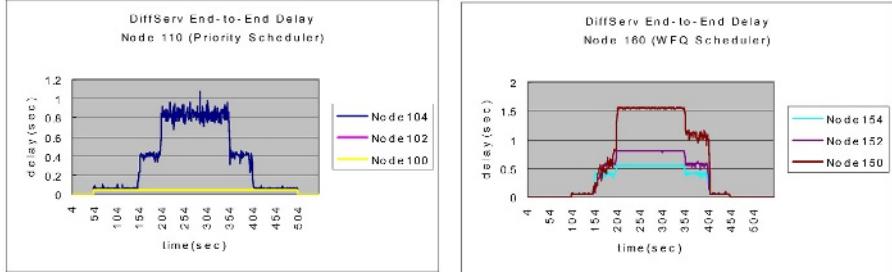
### 3.2 Guaranteed Bandwidth Provisioning

Fig. 3(a) shows the monitored bandwidth utilizations at node 100, 102 and 104 where priority DiffServ packet scheduling is used, and Fig. 3(b) shows the monitored bandwidths at node 150, 152 and 154 where WFQ DiffServ packet scheduling is used. In the simulation, the link between LSR220 and LSR221 is congested by the packet flows between host pairs 200-201 and 202-203.

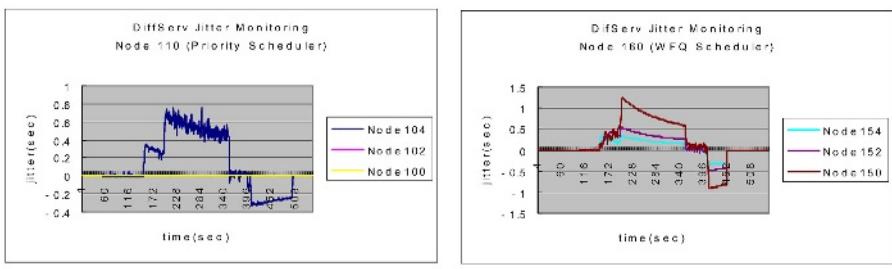


**Fig. 3.** Monitored Bandwidth Utilization.

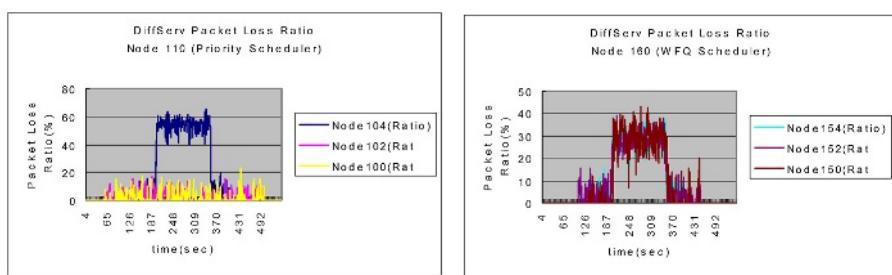
In the priority scheduling, the higher priority packet flows have guaranteed bandwidth at traffic congestion, while the best effort traffic with lower priority is suffering reduced bandwidth at traffic congestion. In WFQ scheduling, however, the overall bandwidth utilization is re-adjusted according to the weight of each DiffServ class-type.



**Fig. 4.** End-to-end Delay.



**Fig. 5.** Jitter.



**Fig. 6.** Packet Loss Ratio.

### 3.3 End-to-End Delay and Jitter

Fig. 4 shows the measured end-to-end delay comparing different packet scheduling, and Fig. 5 shows the jitter. In the priority-based packet scheduling, the higher priority flows are not affected by the congestion, while the end-to-end delay and jitter of the lower priority flow are severely increased at the traffic congestion. In WFQ scheduling, the three packet flows are affected relatively according to the weight.

### 3.4 Packet Loss Ratio

Fig. 6 shows the monitored packet loss ratio with different DiffServ packet scheduling. As we expected, in the priority scheduling, the packet loss rate of the lower-priority packet flow is severely increased at the link congestion, while the higher priority packet flows are not affected by the link congestion. In WFQ scheduling, the packet loss ratios of each packet flows are increased at the link congestion.

### 3.5 Controlled Bandwidth Borrowing with Redistribution of Extra Available Bandwidth

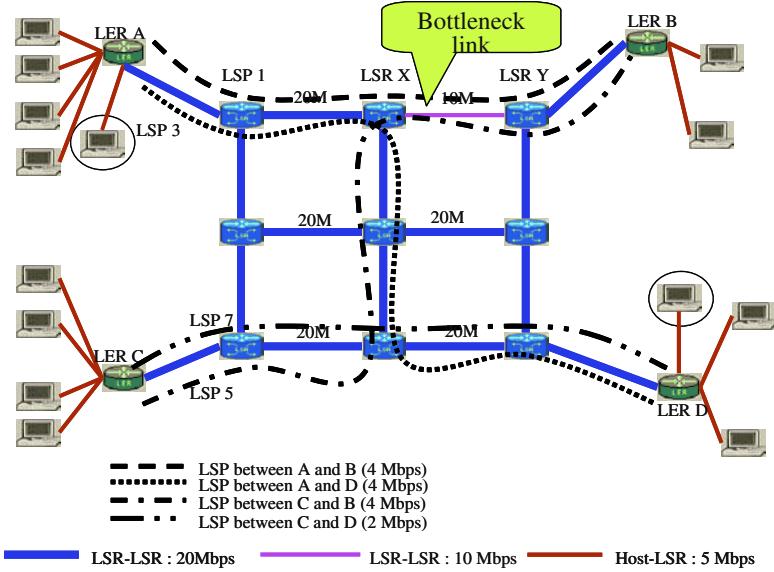
We demonstrate the usefulness of our proposed scheme of dynamic distribution of extra bandwidth using NIST GLASS network simulator [16]. Fig. 7 shows the test network configuration with 9 LSRs, 4 label edge routers (LERs), and 14 hosts. Each link among LSRs has 20Mbps capacity and 0.5msec propagation delay, while the links between hosts and LERs have 5Mbps bandwidth and 1msec propagation delay. To consider the bottleneck link, we reduce the bandwidth of link between LSR X and LSR Y to 10Mbps. In order to deliver user traffic we establish three LSPs of 4Mbps bandwidth, and one LSP between LER C and LER D that has bandwidth of 2Mbps. Each LSP is setup at 25sec and each host starts transmission at 50sec. The controlled bandwidth borrowing mechanism is executed from 170sec in the simulation.

Each host generates mixed UDP and TCP traffic that flows through the four LSPs. We set the target link utilization at 80%, so we cannot establish more LSP or redistribute bandwidth more than that. Each LSR performs a dual-token bucket traffic policing, and core LSRs use the active queue management such as RED.

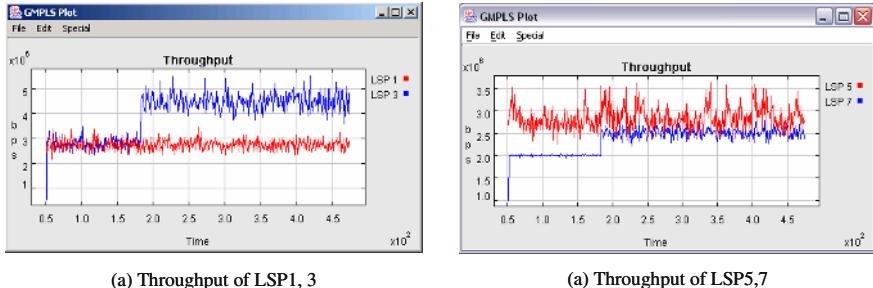
We first verify that the proposed scheme's behavior under the given test network. Because there is bottleneck link between LSR X and LSRY, the LSP 1 and LSP 5 cannot take the redistributed extra bandwidth. However LSP 3 and LSP 7 can get the redistributed bandwidth and achieve high link/LSP utilization.

Fig. 8(a) depicts the throughput of LSP 1 and LSP 3. Since LSP 1 passes through the bottleneck link between LSR X and LSR Y, this LSP cannot take the redistributed extra-available bandwidth along the path. However LSP 3 takes extra-available bandwidth through the proposed scheme. We assign the weight value each LSP to get the extra-bandwidth according to their weight value.

Fig. 8(b) depicts the throughputs of LSP 5 and LSP 7. Since LSP 5 goes through the bottleneck link, it also cannot take the extra-bandwidth. However LSP 7 gets extra-bandwidth through the proposed scheme. Since we assign a different weight value to LSP 7, it gets less extra-bandwidth.



**Fig. 7.** Test Network Configuration for Controlled Bandwidth Borrowing.

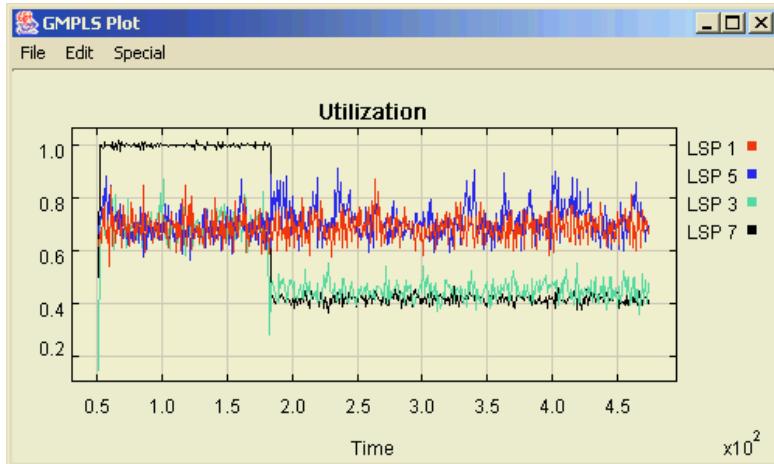


**Fig. 8.** Throughput of LSP 1, 3, and LSP 5, 7.

This simulation shows that we can dynamically re-distribute the extra-available bandwidth to the LSPs that require more bandwidth temporarily and maximize the LSP and link utilization. Fig. 9 depicts the utilization of four LSPs. While LSP 1 and LSP 5 do not get extra-bandwidth, LSP 3 and LSP 7 get extra-bandwidth and utilize their bandwidth.

## 4 Conclusions

In this paper we proposed and analyzed an implementation scheme of the integrated DiffServ-aware-MPLS traffic engineering mechanism for guaranteed QoS and maximized network utilization. In the proposed scheme, each DiffServ class-type is man-



**Fig. 9.** Utilization of four LSPs.

aged with per-class-type policing (metering and marking) and queuing controlled by committed data rate (CDR) and peak data rate (PDR) with related buffer sizes. A hierarchical DiffServ packet scheduler controls the aggregated packet flow of multiple DiffServ class types, where a priority-based scheduling is used for overall traffic and a weight-based scheduling is used for the AF traffic flows. The MPLS LSP packet scheduler is also organized with the hierarchical configuration of weight-based scheduling and priority-based scheduling. The MPLS LSP packet scheduler calculates the extra available bandwidth of the trunk LSP that is un-used by other LSPs.

The performance of the proposed scheme has been analyzed by the NIST GMPLS networking simulator with several different scenarios. From the simulation results of DiffServ-aware-MPLS, we could confirm that the priority-based packet scheduling can be used efficiently to provide premium service in the DiffServ-aware-MPLS structure, while the WFQ scheduling can be used efficiently to provide relative weight mechanism in bandwidth sharing among the constituent class-types in an E-LSP.

From the simulation of the controlled bandwidth borrowing among LSPs, we could verify the applicability of the proposed scheme. The proposed controlled bandwidth borrowing scheme eliminates the risk of bandwidth overbooking and gives a controllable bandwidth sharing mechanism to maximize the bandwidth utilization of tunnel LSPs, according to the dynamic network status. This scheme is also applicable in MPLS-VPN network to give a network operator to use bandwidth more effectively with controlled manner.

As further research works, we are working on the comparison of the queuing model analysis results and the simulation results in order to find the appropriate queuing model of the DiffServ-over-MPLS traffic engineering scheme. By using this queuing model, we can design the traffic engineering parameters, and allocate efficiently the network resources for the guaranteed QoS provisioning. Also the detailed configuration experiments of the operational parameters on real MPLS LSRs (such as Cisco 7200 series MPLS router) must be performed.

## References

1. Xipeng Xiao et al., "Traffic Engineering with MPLS in the Internet," IEEE Network, March/April 2000, pp. 28 ~ 33.
2. IETF RFC 2702, "Requirements for Traffic Engineering over MPLS," Awduche et al., September 1999.
3. IETF RFC 3031, "Multiprotocol Label Switch Architecture," E. Rosen, A. Viswanathan, R. Callon, Jan. 2001.
4. IETF RFC 2676, "QoS Routing Mechanisms and OSPF extensions," G. Apostolopoulos et al., August 1999.
5. IETF RFC 3209, "RSVP-TE: Extensions to RSVP for LSP tunnels," December 2001.
6. IETF RFC 3270, "Mutiprotocol Label Switching (MPLS) support of Differentiated Services," April 2002.
7. IETF Draft, "Constraint-based LSP Setup using LDP," February 2001.
8. IETF RFC 3107, "Carrying Label Information in BGP-4," May 2001.
9. IETF RFC 2475, "An Architecture of Differentiated Services," S. Blake et al., December 1998.
10. IETF Draft, "MPLS Support of Differentiated Services using E-LSP," S. Ganti et. al, April 2001.
11. IETF RFC 2309, "Recommendations on Queue Management and Congestion Avoidance in the Internet," B. Braden et. al, April 1998.
12. IETF RFC 2597, "Assured Forwarding (AF) PHB Group," J. Heinanen et. al, June 1999.
13. IETF draft, "The notion of overbooking and Its Application to IP/MPLS Traffic Engineering," draft-cchen-te-overbooking-01.txt, Informational.
14. Sally Floyd and Van Jacobson, "Link-sharing and Resource Management Models for Packet Network," *IEEE/ACM Trans. on Networking*, Vol. 3 No. 4, August 1995.
15. L. Massoulie and J. Roberts, "Bandwidth sharing: objectives and algorithm," in *Proc. IEEE INFOCOM*, March 1999.
16. NIST GMPLS Simulator – A Scalable Discrete Event Simulator for the GMPLS-based Next Generation Optical Internet, <http://dns.antd.nist.gov/glass/>.
17. SSF (Scalable Simulation Foundation), <http://www.ssfnet.org/exchangePage.html>.
18. J M Pitts, J A Schormans, *Introduction to ATM Design and Performance with Applications Analysis Software*, Section 6.4, John Wiley & Sons, 1996.

# **Architecture Based Software Reengineering Approach for Transforming from Legacy System to Component Based System through Applying Design Patterns**

Jung-Eun Cha, Chul-Hong Kim, and Young-Jong Yang

S/W Engineering Department, Electronics and Telecommunications Research Institute  
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, Korea  
{mary2743, kch, yangyj}@etri.re.kr

**Abstract.** Many reengineering approaches have focused on extracting an abstract representation through syntax analysis of legacy source codes. So, recovery of rationale behind the design decision, such as domain specific semantics and roles, has been ignored. In this paper, we suggest the architecture based reengineering approach using design patterns. A design pattern, as core element of software architecture, has integrated the concept of standardization about certain domain and expert experience into a set of related components that can perform specific functionality with better structure. We describe the reengineering process that defines a architecture of target system by refining architecture information of legacy system extracted through domain analysis, identifies the reengineering patterns that are applicable in that architecture, and completes a target system by mapping the identified reengineering pattern into target architecture. Also, we construct the Servlet2EJB prototyping system transforming servlet programs into EJB components as a case study for realizing our reengineering process.

## **1 Introduction**

A legacy system is an application that was developed on older technology and is past its prime use, but still play an important part in current businesses[1]. Therefore, this legacy system is being viewed as an asset that represents an investment that grows in value rather than a liability whose value depreciated over the times[2]. Now legacy systems are being pressured to concurrently respond to increasing requirements. So, most legacy systems are faced with changing industry models such as e-Business and globalization, changing business models such as CRM, emerging information technologies such as Internet and open system, emerging new information architectures such as J2EE, Web service and component reuse[3]. Also these legacy software systems have suffered from lack of standardization and openness, difficulty of change, and absence of distributed architecture. Instead of continually maintaining these legacy systems at high cost for applying new technologies and extending their business requirements, reengineering them to new systems with good design and architecture can improve their understandability, reusability and maintainability. Therefore, the main issues in current reengineering approaches are how to integrate legacy system with emerging

technologies for reflecting incremented requirements, reuse the legacy assets for producing S/W system in future, and improve the availability and quality of legacy system.

A S/W reengineering is the systematic transformation of existing system into new formed system to realize quality improvements in operation, system capability, functionality, performance or evolution a lower cost, schedule or risk to the customer[4]. Many reengineering approaches have focused on the extracting an abstract representation through syntax analysis of legacy source codes. These approaches suggest some practical problem solutions in discovering a program structure and control flow, but have important shortcoming in understanding the specialized roles and semantics, acquiring the common assets from the specialized domain of legacy system. So, recovery of rationale behind the design decision, such as domain specific semantics and roles, has been ignored. Recently, some attempts towards pattern-based reengineering technologies are tried, but they have some problems such as lack of a systematical reengineering methodology, difficulty of mapping domain knowledge into system elements, and absence of concrete reengineering guidelines.

We use design patterns for reflecting specific knowledge within domain of legacy system to the elements of target system, and to utilize them as the solutions suggested by preceded experts about common problems that must solve in reengineering process.

In this paper, we suggest the architecture based reengineering approach using design patterns. A design pattern, as core element of software architecture, has integrated the concept of standardization about certain domain and expert experience into a set of related components that can perform specific functionality with better structure. We explain the reengineering process that defines a architecture of target system by refining architecture information extracted through domain analysis, identifies the reengineering patterns that are applicable in that architecture, and completes target system by mapping identified reengineering patterns into target architecture. The reengineering patterns that are used in our reengineering process, are based on Gamma patterns[5] re-interpreted from a reengineering viewpoint and, the patterns for reengineering published by Ali Arsamjani[6], Richard Veryard[7], and so on. Also, we explain the Servlet2EJB project transforming servlet programs into EJB components as a case study for realizing our reengineering process. Servlet2EJB is a prototyping system analyzing servlet programs, extracting component information and generating EJB components.

This paper is organized as following. Related works similar to our approach are addressed briefly in section 2. In section 3, concepts of our approach and overall reengineering process are described. In section 4, we present Servlet2EJB project showing case study about our reengineering approach. Lastly, conclusion and future works are addressed.

## 2 Related Works

Reengineering Unified Model II)[8] defined in SEI CMU as a reengineering methodology to be referred most widely. CORUM model is a reengineering tool interopera-

tion to include software architecture concepts and tools. The extended framework - called CORUMII- is organized around the metaphor of a “horseshoe”, where the left-hand side of the horseshoe consists of fact extraction from an existing system, the right hand side consists of development activities, and the bridge between the sides consists of a set of transformations from the old to the new. As another reengineering methodology, there is MORALE(Mission ORiented Architecture Legacy Evolution)[9] that developed in Georgia Institute of Technology. It features an inquiry-based approach to eliciting change requirements, a reverse engineering technique for extracting architectural information from exist code, an approach to impact assessment that determines the extent to which the existing system’s architectural components can be reused in the evolved version, and a specific technique for dealing with the difficulties that arise when evolving user interface.

Especially, the similar approaches to our paper including pattern-based reengineering are [10],[11],[12],[13]. But, they bring an information damage that may happen while maps design decision information to codes because they only emphasize the analysis of legacy codes. Also, as identifying pattern structure from code is very hard, the researches are only considering an extraction of structural pattern of Gamma and are dependent to specific legacy language. Recently, researches that aim at pattern identification of all types(structural, creation, and behavior patterns of Gamma) through UML-based meta-modeling are progressing[14].

### **3 Architecture Based Reengineering Process**

#### **3.1 Common Problems for Reengineering**

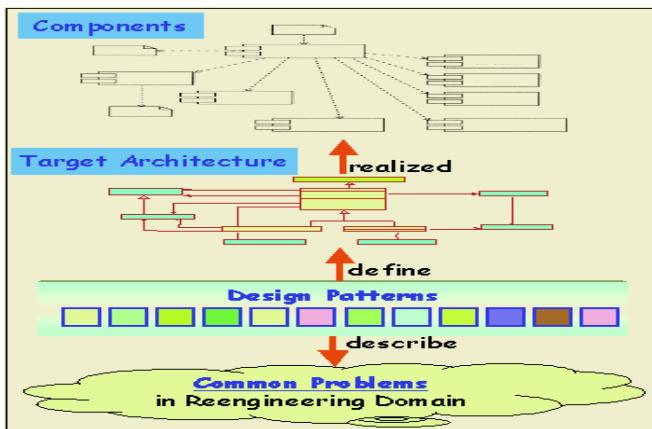
According as web is spread rapidly in most application domain, new technologies such as distributed architecture and component technology are required to support this change. Because of changes of business model such as e-Business, and appearance of new platforms such like Web service, we may consider all systems as legacy systems, and need to collect common requirements for reengineering of legacy system to reuse a legacy system as asset in organization. We classified the common problems that must solve during reengineering process like a <Table 1>.

#### **3.2 Core Concepts**

Design patterns for reengineering can be utilized as core assets to understand the elements of legacy system. Also, they are building block of related classes used to compose target architecture, and the expert’s solutions about common problems must solve in reengineering process. Therefore, they are used in creating the architecture of target system because they can assist legacy systems to evolve to component based system by defining design patterns as a package unit that is set of components that achieve specific functionality for reengineering. (Figure 1) represents our paper’s basic concepts. We can recover a general meaning from legacy system by using reengineering patterns as reusable element. This approach allow to assist legacy systems to evolve to component based system seamlessly by defining design patterns as a

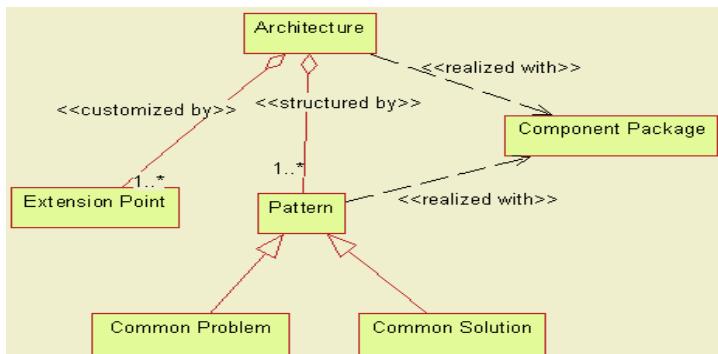
**Table 1.** Common problems that must solve for reengineering.

Types	Problem Contexts
<b>Business Modeling &amp; Requirements</b>	<ul style="list-style-type: none"> <li>• Domain analysis of legacy system</li> <li>• Recovery of business process</li> <li>• Establishment of reengineering goals and strategies</li> </ul>
<b>Legacy Understanding</b>	<ul style="list-style-type: none"> <li>• Finding of problems found in legacy system</li> <li>• Understanding of legacy information which follow to viewpoints(function/structure/action/quality items)</li> <li>• Architecture Recovery</li> <li>• Recovery of specific context and limitations of legacy system</li> </ul>
<b>Information Refinement</b>	<ul style="list-style-type: none"> <li>• Restructuring</li> <li>• Documentation and inventory of recovery information</li> <li>• Module identification and component extraction</li> <li>• Definition of system architecture</li> <li>• Definition of reengineering guidelines</li> </ul>
<b>Implementation</b>	<ul style="list-style-type: none"> <li>• Definition and implementation of component</li> <li>• Interface wrapping</li> <li>• Implementation of interface with heterogeneous platforms</li> <li>• Implementation of migration adaptors integration components into target system</li> </ul>
<b>Testing</b>	<ul style="list-style-type: none"> <li>• Definition of estimation elements</li> <li>• Optimization of target architecture</li> <li>• Unit testing and integrating testing of components</li> <li>• Analysis of expectation effective</li> </ul>

**Fig. 1.** Basic Concept of Pattern Based Reengineering.

package unit that is set of components that achieve specific functionality for reengineering and then, design patterns are used in creating architecture of target system.

(Figure 2) is a conceptual metamodel of architecture based target system constructed by S/W reengineering. “Architecture” of target system includes several patterns and is structured by relationships between these patterns. “Pattern” is an element in architecture and contains common problem and the solution that can happen in specific domain for reengineering. “Component Package” that performs specific functionalities realizes a pattern and architecture. Architecture includes “Extension



**Fig. 2.** One Basic Concept of Pattern Based Reengineering.

Point" so that each pattern may can integrate and accommodate new reengineering requests to remainder parts of the target system.

### 3.3 Reengineering Design Pattern

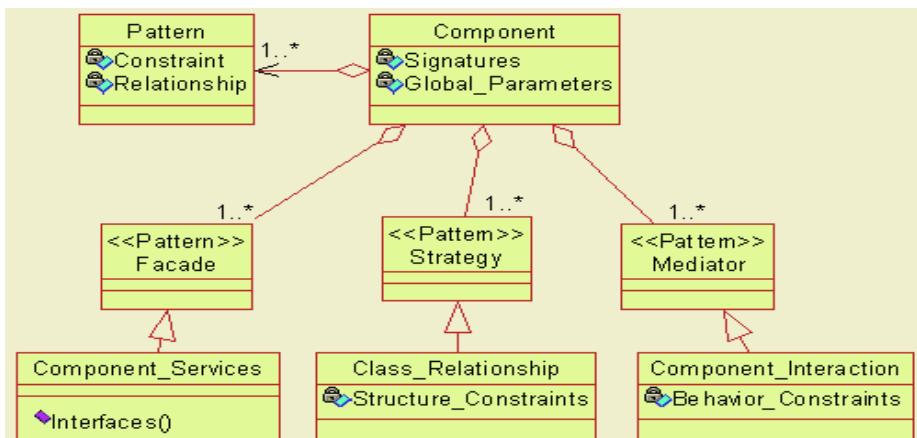
Reengineering design patterns that describe solutions about common requirements of reengineering that presented in <table 1> already exist in the form of specific contexts, so we can apply them very usefully in our reengineering process. They include the patterns that are defined newly as solutions for reengineering requirements listed in <table 1> as well as Gamma's patterns that have been applied widely as general design principles in forward engineering process.

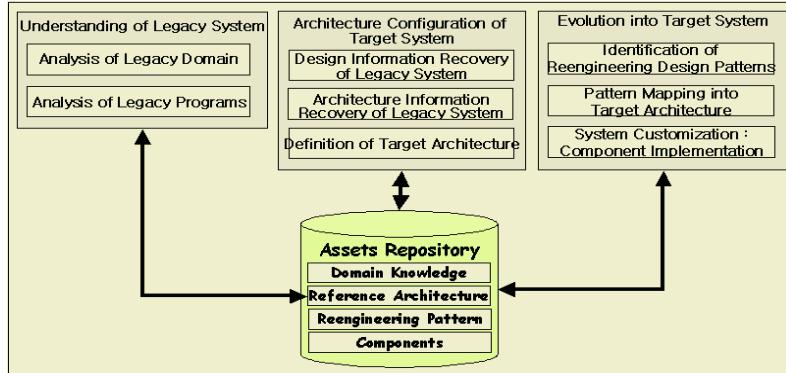
We present the reengineering patterns as <table 2>. Patterns for reengineering that defined by Ali Arsamjani[6] or Richard Veryard[7] etc., have been published continuously in journals of S/W Engineering. Now, pattern integration technologies for customizing patterns that are selected by reengineering stakeholders from existing reengineering patterns are required. (Figure 3) is shown the structure of design pattern that is consisted of components with interaction relationships. We can explain an interaction scenario between components for achieving a specific purpose through this structure. Among the Gamma's patterns, we used the Bridge pattern, the Factory Method pattern, the Adapter pattern, and the Mediator pattern to explain interface based independent links between components mutually, as temple of pattern structure.

Patterns are consisted of several components that have pattern signatures and global variables between related components. Components provide an integrated abstract interface by Facade pattern. Also, pattern includes Component\_Relationship class that express structural restriction condition and Component\_Interaction class that express limitation about dynamic integration in application domain, and each classes are embodied their features through Strategy and Mediator pattern each.

**Table 2.** Reengineering Design Patterns.

Purpose	Characteristic description	Candidate pattern
GUI interacting design	The complicate interactions should be eliminated with the isolation between GUI and the processing	Mediator
Independent separation of modules	To lower the coupling within a complex system which consists of sequential sub-functions	Mediator, Command
State Control	To control and monitor the transitions state of processes and the whole system	State
Module Hierarchy	To lower the coupling within a system which consists of a set of structural sub-functions	Mediator, Facade
Interface of process	Supply an accordant interface of process	Template
Wrapping	Supply a wrapper to hide the detail of implementation and location of objects from the client	Decorator, Adapter, Proxy
Execution of dynamic objects	Encapsulating the processing algorithms for requests of run-time re-configurable objects	Builder, Factory Method
Componentization	Construction of reusable component from legacy business logic[Logic Extraction]	Mixing of Composite and Facade
Process Migration	Transmission logics of legacy system to processing layer in 3-tier architecture and connection establishment with storage layer	Mixing of Proxy and adapter
Reflection of domain features	Modeling domain embraced legacy system family and supplying architecture to understand and utilize a legacy information	Mixing of Abstract Factory & Prototype
Customization of requests	To define hotspots in legacy system and rebuild them by independent algorithm	Mixing of Abstract Factory & Strategy

**Fig. 3.** Pattern Structure.

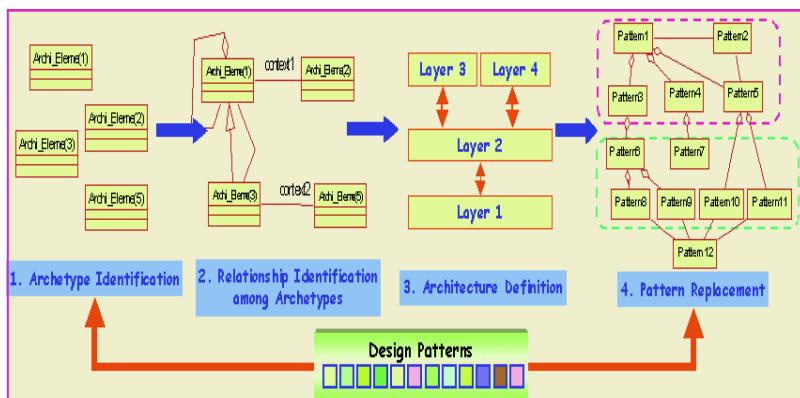


**Fig. 4.** Overall reengineering process.

### 3.4 Reengineering Process

The first step of our reengineering process is to compose the architecture of target system from the design information of legacy system that is recovered through understanding of various reference resources including domain analysis. And through identifying and mapping reengineering design patterns, which are sets of components that embody target architecture, a reengineering process is completed. So, overall reengineering process suggested in this paper is consisted to 3 parts, and they are summarized in (Figure 4) and main activities that are performed at each step of process are explained in <table 4> shortly.

Especially, (Figure 5) shows the conceptual procedures for defining target architecture in reengineering process described in <Table 3>. Firstly, we identify a “Achitype” for generalizing the problems of legacy system and outlining a target system.



**Fig. 5.** Procedures for defining target architecture.

**Table 3.** Reengineering Design Patterns.

<b>Activity</b>	<b>Description</b>
<b>Phase 1</b>	<b>Understanding of Legacy System</b>
<b>Domain Analysis of Legacy</b>	<p>Grasp knowledge given in legacy system development and abstract all legacy information to utilize them in reengineering process</p> <ul style="list-style-type: none"> <li>- Collect know-how, experiences, and knowledge from developers and operators of legacy system</li> <li>- Grasp system reference model by processing type</li> <li>- Grasp core issues, major functionalities, restrictions and main characteristics of legacy system</li> <li>- Grasp common characteristics of domain and information about similar system</li> </ul>
<b>Program Analysis of Legacy</b>	<p>Acquire understanding information through analyzing source codes of legacy system</p> <ul style="list-style-type: none"> <li>- Code restructuring</li> <li>- Lexical analysis and creation of Abstract Syntax Tree</li> <li>- Analysis of call relationships, data and control flow between unit modules</li> <li>- Analysis of semantics in system level such as, system resource, program call etc</li> </ul>
<b>Phase 2</b>	<b>Architecture Configuration of Target System</b>
<b>Design Info. Recovery of Legacy</b>	<p>Prepare basis information for architecture understanding by extracting analysis and design information of legacy</p> <ul style="list-style-type: none"> <li>- Static structure understanding by extracting data structure</li> <li>- Abstraction of functional information by analyzing the use cases and sequence of major screens</li> <li>- Extraction of information about process and data flow by generating activity diagram</li> </ul>
<b>Architecture Info. Recovery of Legacy</b>	<p>Grasp architecture elements(quality/functionality) and identify architecture model focusing on sub-systems</p> <ul style="list-style-type: none"> <li>- Refinement of functional/quality elements from legacy</li> <li>- Identification of subsystems performing independent tasks and analysis of relationship among them</li> <li>- Definition of hierarchy architecture for subsystems</li> </ul>
<b>Definition of Target Architecture</b>	<p>Packages extracted information into as components that have high dependency semantically, and define target architecture</p> <ul style="list-style-type: none"> <li>- Identification of candidate components and analysis of relationship among them</li> <li>- Introduction of reference architecture</li> <li>- Definition of app. architecture for target system</li> <li>- Definition of technical environments and hierarch structure of application component</li> </ul>
<b>Phase 3</b>	<b>Evolution into Target System</b>
<b>Identification of Reengineering Design Patterns</b>	<p>Identification of reengineering patterns to embody application architecture of target system</p> <ul style="list-style-type: none"> <li>- Analysis of requirements for target architecture</li> <li>- Identification and classification of proper reengineering patterns in target architecture</li> <li>- Identification of component that compose each reengineering pattern</li> </ul>
<b>Pattern Mapping into Target Architecture</b>	<p>Map the architecture elements that is semantic package unit into patterns that is a set of components</p> <ul style="list-style-type: none"> <li>- Mapping the architecture structure into reengineering patterns that can embody target architecture</li> <li>- Integrating and division of patterns</li> </ul>
<b>System Customization: Component Implementation</b>	<p>Replace the target architecture by components that are packaged in patterns, and evolve target system into system</p> <ul style="list-style-type: none"> <li>- Refinement component relationships Replacement each pattern by interaction structure of components</li> <li>- Extension target architecture that forms pattern structure to component structure</li> </ul>

It is a functional unit which occurs over and over again in target system and a set of entities for addressing core parts in system. So, Archetype is a very stable unit. Next, we define the relationships among identified Archetypes. Because these relationships represent not only domain specific meaning but also the flows of data and control, they can form design patterns to be applicable in construction of practice target system. And then, we define a hierarchical S/W architecture of target system by integrating and re-dividing the identified Archetypes based on their relationships[15].

## 4 Case Study

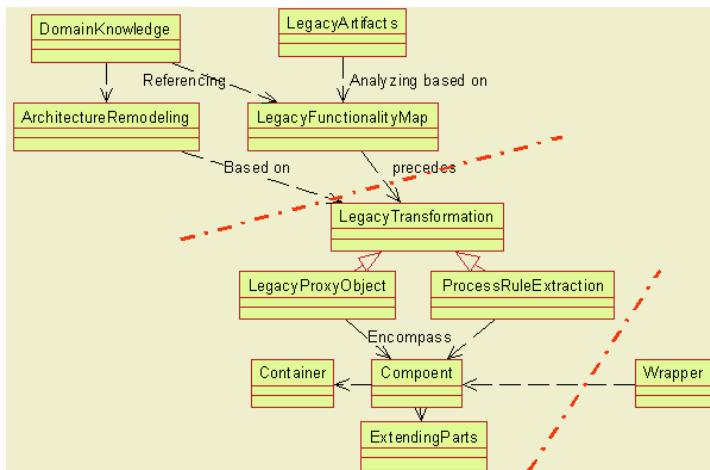
### 4.1 Requirements for Servlet2EJB

We constructed the Servlet2EJB project that transforms servlet programs into EJB components in J2EE architecture for applying the concepts and process suggested in this paper. Generally, servlet programs are written in various languages such as Java module, HTML, and SQL etc., and one servlet page performs one independent business processing. So, parsers for various languages should analyze servlet programs. The analyzed information is classified according to characteristics of EJB component and redefined according to EJB specification. Reengineering requirements for Servlet2EJB are summarized as following:

- Analyze the files in input project, and then store the results to DB, and reconfigure them so that may can use in relevant during reengineering process. Also, support multiple languages analysis such as script, HTML, Java, SQL etc.
- For extracting the EJB components, cluster the analyzed legacy information and redefine as to EJB specification. To do this, analyze servlet logic code from servlet files and, generate information for transformation into each bean type (Session/Entity). Also, integrate the SQL analysis information for Entity bean creation.
- Create bean classes, interface and DD(Deployment Description ) and bind them into standard jar file.

### 4.2 Reference Architecture

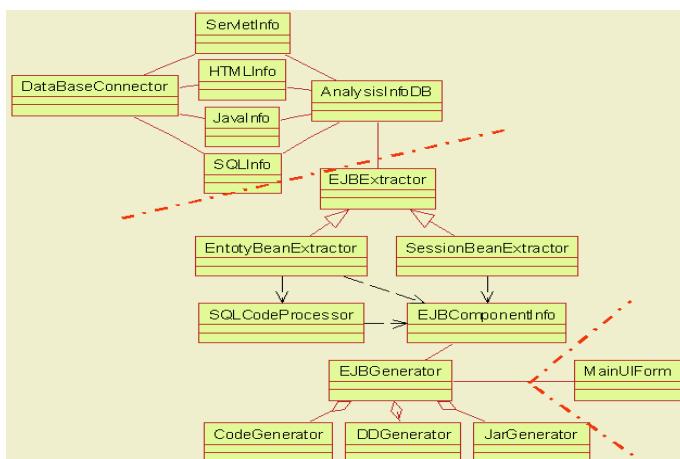
Simplified system architecture that is required generally for reengineering legacy system is like as (figure 6). This reference architecture is structured to three parts that are divided by dotted line, based on 3-tier model of MVC (Model-View-Control) architecture. In first part, we identify the core functionalities of legacy system through analyzing the domain information, and derive new target architecture from the extracted design information of legacy system. In processing layer, the middle part, we extract business processes of legacy system, transform them to EJB specifications, and package a set of components that is independent module. Component is constructed over a new container, or is modified through extension point according to requirements of system. Finally, wrapper supplies transparent access points to outside interfaces.



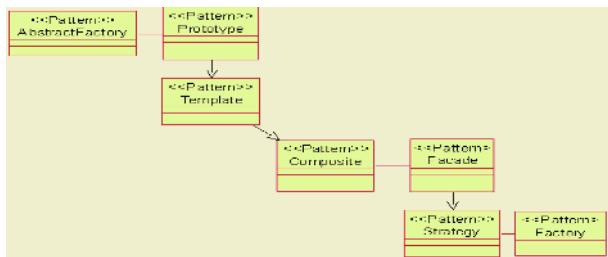
**Fig. 6.** Simplified reference architecture for reengineering of Legacy system.

#### 4.3 Transformation Servlet Programs into EJB Components

The architecture of target system transforming servlet programs to EJB component is like a (figure 7). Top part in figure has the roles for analyzing the servlet programs and identifying core modules in architecture of reengineering system defined in (figure 6). In the middle part, the business logics for EJB components are extracted, and then EJB component specifications are created finally. In last part, a form of interface is created to access the components of target system.

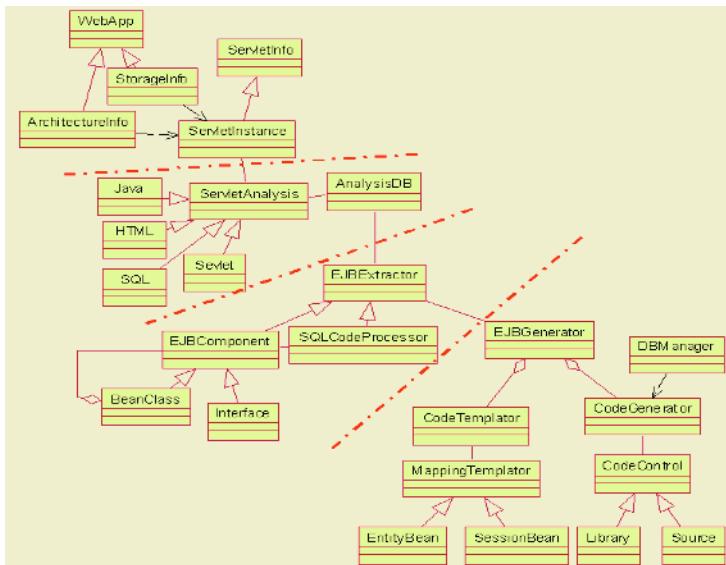


**Fig. 7.** Architecture of target system transforming servlet programs to EJB.

**Fig. 8.** Pattern-Based Architecture for Servlet2EJB.

(Figure 8) is pattern-based architecture for the Servlet2EJB modeled by mapping the patterns selected from <table 2> to target architecture in (figure 6). The Abstract Factory and Template patterns are applied to acquire family information related to domain and to extract various type of legacy information. Template pattern is used to analyze business information of various language included in servlet and to store these information that can be accessed in transparent way. Also, Composite and Facade patterns are used for transforming the various types of the legacy systems by applying a unique form and used for providing a unified interface. And Strategy pattern is used to prepare various rules for code generation according to EJB component types and is used to apply these rules as independent algorithm in generating real codes. The target system for Servlet2EJB could be supported the guidelines from design decision of individual pattern offered in the pattern structure. Also, could develop classes of target system by structure that is coincided in pattern.

(Figure 9) is class diagram for target system generated by extending pattern structure presented in (figure 8).

**Fig. 9.** Class diagram of Servlet2EJB.

## 5 Conclusion

We have considered a design pattern as the best means for reflecting specific knowledge existed in domain of legacy system to elements of target system in concrete form, and utilizing the solutions suggested by the experts about common problems that must be solved in reengineering process.

The similar approaches to our paper including pattern-based reengineering have suggested in [10],[11],[12],[13]. But, these researches have some troubles such as information loss that may happen during mapping design decision information to codes because they only emphasize the analysis of legacy codes. Also, because identifying pattern structure from codes is very hard, these researches are only considering extraction of structural pattern of Gamma, and they are dependent to specific legacy language.

So, in this paper, we have presented the architecture based reengineering approach using design patterns. Design pattern, as core element of software architecture, has integrated the concept of standardization about certain domain and expert experience into a set of related components that can perform specific functionality with better structure. We describe a reengineering process that defines a target architecture by refining architecture information extracted through domain analysis, identifies the reengineering patterns that are applicable in that architecture, and completes target system by mapping identified reengineering pattern into target architecture. Also, we explain the Servlet2EJB project transforming servlet programs into EJB components as a case study for realizing our reengineering process. Servlet2EJB is prototyping system analyzing servlet programs, extracting component information and generating EJB components.

Now, we have suffered from collecting reengineering patterns and constructing a library for them. Also, we are in difficulties for lack of systematical procedures and guideline to evolve our approach. In future, we plan to establish formal algorithm for extracting reengineering patterns from legacy source codes, and related documentations. And we will construct a supporting tool including sub-systems and repository to manage components, patterns and architecture that can be reused during reengineering process.

## References

1. Dolly M, Neumann : Evolution Process for Legacy System Transformation. IEEE Technical Applications Conference, Northcon'96(1996), pp57-62
2. Nelson Weideman, Dennis Smith, Scott Tilley : Approaches to Legacy System Evolution. CMU/SEI-97-TR-014,<http://www.sei.cmu.edu/pub/documents/97.reports/pdf/97tr014.pdf>
3. William Ulrich : Legacy Systems - Transformation Strategies. Prentice Hall(2002)
4. SEI Reengineering Center : Perspectives on Legacy System Reengineering.(1995) <http://www.sei.cmu.edu/reengineering/pubs/lseesree/lseesree.pdf>
5. Eric Gamma : Design Patterns:Elements of Reusable Object Oriented Software. Addison-Wesley(1994)

6. Ali Arsanjani : Component-based Development and Integration Pattern Language. PLoP conference 2000)
7. CBDi Forum : Pattern Catalog. <http://www.cbdiforum.com/patterns/index.php3>
8. Rick Kazman, Steven G. Woods, S. Jeromy Carriere : Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. Fifth Working Conference on Reverse Engineering, October(1998) pp: 154-163
9. Abowd G. Goel A. Jerding D.F., McCracken M., Moore M., Murdock J.W., Potts C., Rugaber S., Wills L. : MORALE. Mission ORiented Architectural Legacy Evolution. International Conference on Software Maintenance(1997) pp: 150 -159
10. G Antoniol, R. Fiutem, Cristoforetti : Design Pattern Recovery in Object Oriented Software. 6th Workshop on Program Comprehension(1998)
11. Christian Kramer, Lutz Prechelt : Design Recovery by Automated Search for Structural Design Patterns in Object-Oriented Software. 3rd Working Conference on Reverse Engineering(WCRE: 1996) pp:208-215
12. Rudolf K. Keller, Reinhard Schauer, Sébastien Robitaille, and Patrick Pagé : Pattern-Based Reverse-Engineering of Design Components. Conference on Software Engineering (ICSE : 1999) pp:226-235
13. William C. Chu, Chih-Wei Lu, Chih-Peng Shiu, Xudong He : Pattern Based Software Re-engineering : A Case Study. Journal of Software Maintenance: Research and Practice, Vol. 12, No. 3, May/June(2000) pp: 300-308
14. Jochen Seemann, Jürgen Wolff von Gudenberg : Pattern-Based Design Recovery of Java Software. Communications of the ACM, Vol. 38, No. 10, pp:65-74, October(1995)
15. Jan Bosch : Design and Use of Software Architectures. Addison-Wesley, May(2000)

# Increasing the Efficiency of Cooperation among Agents by Sharing Actions

Kazunori Iwata<sup>1</sup>, Mayumi Miyazaki<sup>2</sup>, Nobuhiro Ito<sup>3</sup>, and Naohiro Ishii<sup>4</sup>

<sup>1</sup> Dept. of Business Administration, Aichi University

370, Kurozasa, Miyoshi-cho, Nishikamo-gun, Aichi, 470-0296, Japan

Phone: +81-561-36-1119 (Ext. 8802), Fax: +81-561-36-5546  
kazunori@vega.aichi-u.ac.jp

<sup>2</sup> Dept. of Intelligence and Computer Science, Nagoya Institute of Technology

miyazaki@egg.ics.nitech.ac.jp

<sup>3</sup> Dept. of Computer Science and Engineering, Graduate School of Engineering, Nagoya Institute of Technology

Gokiso-cho, Showa-ku, Nagoya, Aichi, 466-8555, Japan

Phone: +81-52-735-5567, Fax: +81-52-735-5567  
bobson@phaser.elcom.nitech.ac.jp

<sup>4</sup> Dept. of Information and Network Engineering, Aichi Institute of Technology,  
1247, Yachigusa, Yagusa-cho, Toyota, Japan, 470-0392

Phone: +81-565-48-8121 (Ext. 2801), Fax: +81-565-48-0070  
ishii@in.aitech.ac.jp

**Abstract.** Recently, multi-agent techniques are being studied with keen interest to develop flexible information systems for complicated problems. In particular, rescue systems for reducing the damage caused by serious disasters are of international interest. The RoboCupRescue Project[1], launched in Japan in 1999, is an international research project aimed at developing a rescue system for serious disasters. In this paper, we focus on the system controlling the rescue agents. One aspect of great interest regarding multi-agent techniques is the design of systems that realize efficient cooperative behavior. Traditional frameworks for cooperative behavior design include cooperative protocols, hierarchical approaches and so on. However, autonomous robot agents such as rescue agents can not communicate with each other and form groups to cooperate within these traditional frameworks, because these agents have limited mobility and can obtain only restricted information. Moreover, they need to behave in a real-time environment which is complicated and changing continuously.

In this paper, we propose a new cooperative model for multi-agent systems which improves the performance of the systems by reducing redundant actions among the agents. Moreover, the results of disaster simulations confirm that our model is efficient for such complicated environments.

## 1 Introduction

Remarkable progress has been made in the fields of artificial intelligence and robotics techniques facilitating the development of flexible systems for environments which change continuously and in a complicated fashion – in particular, rescue systems for

serious disasters aimed at decreasing the damage caused by the disaster and saving lives. To realize these systems, multi-agent approaches are being studied with great interest. In multi-agent systems, certain agents can solve complicated problems by behaving both autonomously and cooperatively. Therefore, multi-agent techniques make the development of flexible systems possible. The RoboCupRescue Project started in 1999 in Japan, and is an international research project aimed at developing a rescue system for serious disasters. This project uses multi-agent approaches to realize the system, which we focus on in this paper.

One aspect of great interest regarding multi-agent techniques is the design of multi-agent systems that realize efficient cooperative behavior. Traditional frameworks for cooperative behavior design include cooperative protocols, hierarchical approaches and so on [2,3,4,5,6,7,8].

However, autonomous robot agents such as rescue agents can not communicate with each other and form groups to cooperate within these traditional frameworks, because these agents have limited mobility and can obtain only restricted information. Moreover, they need to behave in a real-time environment which is complicated and changing continuously. For example, we do not know when and where a fire may break out. The agents should behave both autonomously and cooperatively to extinguish the fire according to the conditions, even if another fire breaks out at the same time but in a different location. To realize the complicated cooperative behavior for the agents, we need to design a new cooperative model.

In this paper, we propose a new cooperative model for multi-agent systems. The model improves the performance of the systems by reducing redundant actions among the agents and efficiently solves the problems of the systems. We performed a number of disaster simulations which confirm that our model is efficient for these type of complicated environments.

## 2 Target Systems

In this paper, our research focuses on a complicated multi-agent system such as a disaster simulation. It is important for our system to model a real disaster as accurately and realistically as possible. In this paper, our multi-agent system for disaster simulation contains the following conditions:

1. Problems may occur simultaneously.
2. The nature of a problem may change over time.
3. Problems which an agent cannot solve by itself may occur.
4. The total number of agents in the system is unknown.

These conditions are closely related to each other.

## 3 Forming Groups and Sharing Actions in a Group

Autonomous robot agents such as rescue agents can form groups to solve problems that are beyond the ability of a single agent. However, some agents may perform redundant actions which have already been completed by other agents in the same group, as all agents would attempt to perform the same set of actions if they have the same goal.

For example, if three agents simultaneously attempt to extinguish a fire which could be extinguished by two agents, the actions of one agent are redundant. Therefore, the agents should perform the fewest number of actions required to efficiently achieve their goal. This condition is caused by the fact that agents do not focus on their actions as part of the group, but rather all agents focus only on their goals. We define a cooperative model in which agents focus on their actions as part of the group and reduce the number of redundant actions by sharing common actions.

### 3.1 Similarity of Goals

Agents decide what their actions should be according to their goals. Therefore, we define a similarity measure for goals, which the agents can use to effectively form their groups. The simplest way to form a group and to share actions is to form a group consisting of agents who have the same goal. However, this condition is so rigid that the agents cannot form many groups. In order to ease the condition, we define a similarity measure for the goals.

Before defining the similarity of goals, we explain what a goal means in this paper. A goal is the indicator to introduce actions using the information around an agent. In other words, an agent uses the information of its environment to select actions according to a goal.

#### **Definition 1. Agents' Goal**

*A goal derives actions from information.*

*Goal :  $I \rightarrow A$*

*where “ $I$ ” denotes the information, including the environmental information, around the agent and “ $A$ ” denotes the action set.*

We define the similarity of a set of goals as the relation among the goals. The similarity of goals having several common actions is defined as the result of derivations.

#### **Definition 2. Similarity of Goals**

*The similarity of goals between  $G_0(I_0)$  and  $G_i(I_i)$  ( $1 \leq i \leq n$ ) is defined as follows:*

*Similarity\_of\_Goals( $G_0, I_0, (G_1, I_1, \dots, G_n, I_n)$ )*

$$= \begin{cases} \text{True} & \text{If } \sum_{i=0}^n |G_0(I_0) \cap G_i(I_i)| > \theta \\ \text{False} & \text{otherwise} \end{cases}$$

*where “ $G_k(I_k)$ ” denotes the action set derived from the goal “ $G_k$ ” in Definition 1 with the information “ $I_k$  ( $0 \leq k \leq n$ )”, “ $|G_0(I_0) \cap G_i(I_i)|$ ” denotes the number of common actions between “ $G_0(I_0)$ ” and “ $G_i(I_i)$ ” and “ $\theta$ ” denotes a threshold.*

If  $n$  equals 2, the function *Similarity\_of\_Goals* is as follows:

*Similarity\_of\_Goals*( $G_0, I_0, (G_1, I_1)$ )

$$= \begin{cases} \text{True } If & |G_0(I_0) \cap G_1(I_1)| > \theta \\ \text{False otherwise} & \end{cases}$$

An agent communicates with other agents to form a group in which all the agents will have the same or similar goals. If an agent communicates directly with all other agents in the system, the amount of communication among agents proliferates. However, an agent only needs to communicate with surrounding agents, because the probability that the surrounding agents have common actions is high. For instance, if an agent would like to extinguish a fire at a particular location and attempts to cooperate with other agents who are far in the distance, they will be unable to find the fire or to move near the original agent to cooperate with it. Hence, any communication between them is a waste of time.

Taking these previous conditions into consideration, we simplify the similarity of goals in Definition 2, because agents which are near each other have similar environmental information. The definition of the simplified similarity of goals is as follows:

### Definition 3. Simplified Similarity of Goals

The similarity of goals between  $G_0(I_0)$  and  $G_i(I_i)$  ( $1 \leq i \leq n$ ) is as follows:

*Simplified\_Similarity\_of\_Goals*( $G_0, I_0, (G_1, \dots, G_n)$ )

$$= \begin{cases} \text{True } If & \sum_{i=0}^n |G_0(I_0) \cap G_i(I_0)| > \theta \\ \text{False otherwise} & \end{cases}$$

where each agent uses only the information “ $I_0$ ” due to the previous considerations.

If  $n$  equals 2, the function

*Simplified\_Similarity\_of\_Goals* is as follows:

*Simplified\_Similarity\_of\_Goals*( $G_0, I_0, (G_1)$ )

$$= \begin{cases} \text{True } If & |G_0(I_0) \cap G_1(I_0)| > \theta \\ \text{False otherwise} & \end{cases}$$

## 3.2 Forming Groups

An agent forms a group with surrounding agents and shares common actions, as explained above. Hence, we can define a model for sharing the actions within a group. The agents in our model can be in one of the following five states during the process of forming a group:

**Proposal State:** The agent who proposes forming a new group is in the “Proposal State”.

**Proposed State:** An agent who receives propositions from other agents is in the “Proposed State”.

**Leader:** The agent who successfully forms a group is the “Leader” of the group.

**Member:** An agent who belongs to a group is a “Member” of the group. Strictly speaking, the leader of the group is also a member, however we distinguish between “Leader” and “Member”.

**Normal State:** An agent who is not in any of the above states is said to be in the “Normal State”.

Initially, each agent is in the “Normal State” and will search for other agents when problems are encountered. When an agent finds other agents and wants to form a group, it sends surrounding agents a message to propose the forming of a group along with its goal and changes its state into the “Proposal State”. If it receives propositions from other agents before sending the proposition, it changes its state into the “Proposed State” and checks the received goals.

An agent which is in the “Proposed State” sends the proposing agent which has the most similar goal(as in Definition 3) a message to indicate it wishes to belong to the group, along with its goal, and sends all other proposing agents a message indicating it declines belonging to their groups. If it cannot find a similar goal, it will send all proposing agents a message declining membership in their groups, it will change its state back into the “Normal State”, and will search for new problems.

When an agent in the “Proposal State” receives the messages and goals from the agents it has contacted, it changes its state into the “Leader” state, forms a new group and sends all agents in the group a message indicating that a new group is being formed. If it receives only messages which decline membership in the group, it will change its state back into the “Normal State” and start solving the problem on its own. An agent which has the state “Leader” then distributes the actions derived from the goals amongst the agents belonging to its group. When it judges the problem to be successfully completed, it sends the members of the group a message indicating that the group is being dispersed, changes its state back into the “Normal State” and begins searching for new problems.

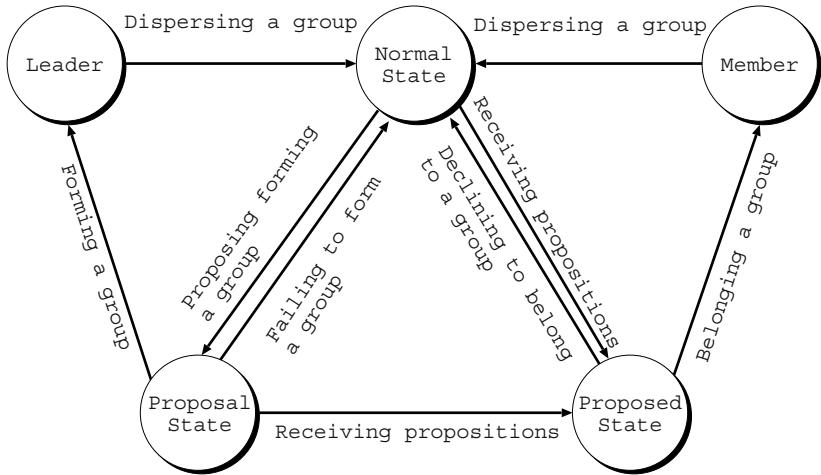
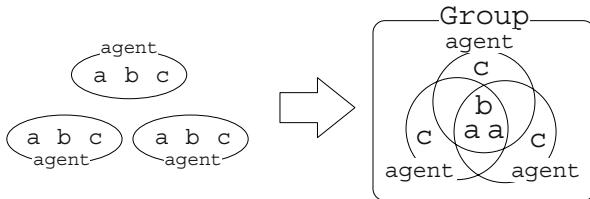
When an agent in the “Proposed State” receives the message indicating that a new group is being formed, it changes its state into the “Member” state and waits for the next messages from the leader. When it receives the message indicating the group is being dispersed, it changes its state back into the “Normal State” and searches for new problems.

The transition of the states is illustrated in Figure 1.

### 3.3 Cooperation and Sharing Actions in a Group

Agents form groups in our model according to the procedure outlined in Subsection 3.2. The agents in one group cooperate with and share common actions with each other to solve problems, and in this subsection we explain how this is done. In our model, a combination of the following techniques is used:

**Cooperation:** The agent who is in the “Leader” state distributes the same actions to agents in its group to solve a problem, if it cannot be solved by only one agent. Hence, these agents cooperate with each other because they receive and perform the same actions.

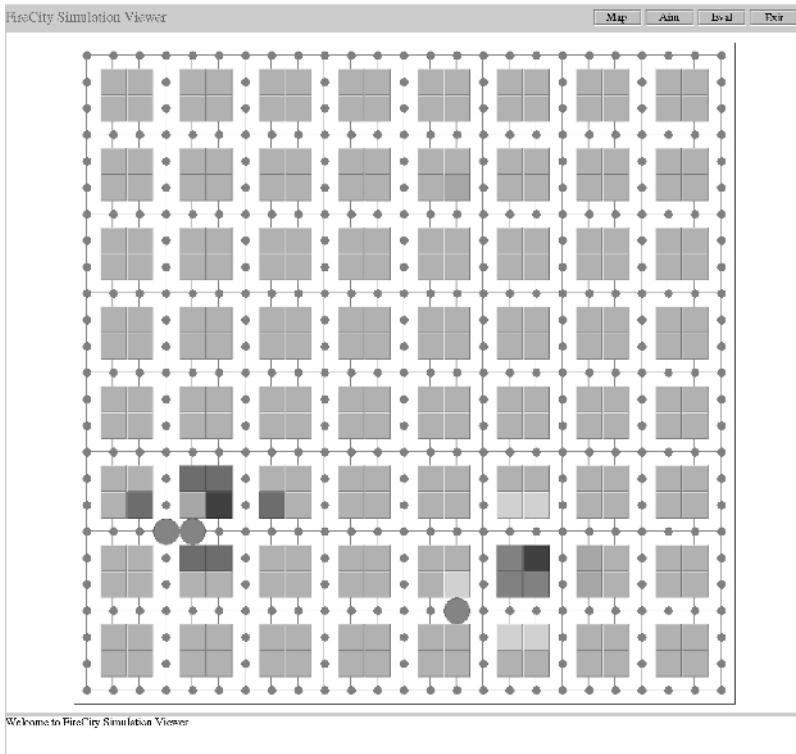
**Fig. 1.** Agents' States**Fig. 2.** Example of Forming Group

**Sharing Actions:** The agent who is in the “Leader” state distributes different actions to the agents in its group, even if their preference would be to perform the same actions. Under this condition, the agents share common actions because they do not all perform the same actions. As a result of this sharing, the total number of actions is reduced.

For instance, if there are three agents and each agent would like to perform three actions named “a”, “b” and “c”, the total number of actions performed would be nine. However, if two actions named “a”, one action named “b” and three actions named “c” are sufficient to solve the problem under the conditions, the agents should cooperate with actions “a” and “c”, and share actions “a” and “b”. This cooperation and sharing means the agents would perform only six actions in total. This example is illustrated in Figure 2.

## 4 Simulation

In this paper, we implemented fire-brigade agents based on our agent model. To evaluate our agent model, we used a simple rescue simulator to determine whether the fire-

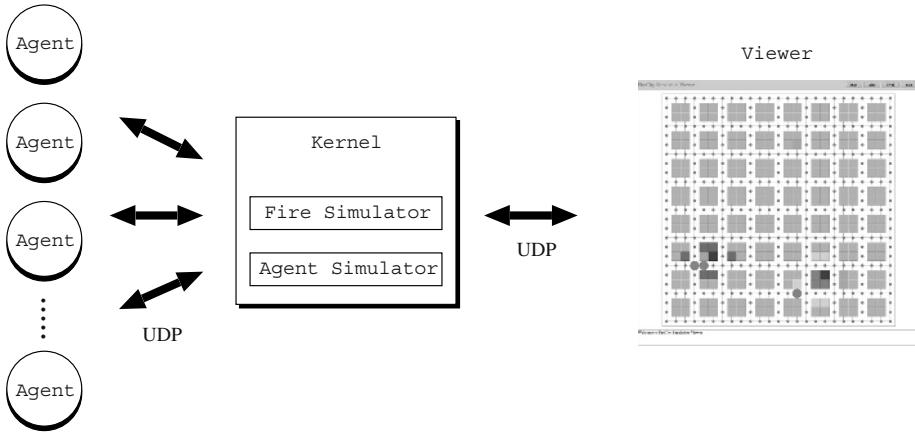


**Fig. 3.** Simple Rescue Simulator

brigade agents could efficiently extinguish a fire within a virtual city. The layout of the virtual city seen in the simulation viewer is illustrated in Figure 3. The RoboCupRescue disaster simulator includes many types of simulators(fire simulator, collapse simulator, traffic simulator and so on) and several different types of agents(police, fire-brigade, ambulance). Moreover, this simulator uses a real city map such as that of Kobe city, Japan. Therefore, we use a simpler rescue simulator which includes only a fire simulator and a simple virtual map[9]. Our simulator consists of a number of fire-brigade agents, a kernel and a simulation viewer. The kernel simulates the burning of buildings and the agents' actions, while the viewer displays the simulation. The agents communicate with each other through the kernel using UDP(User Datagram Protocol) and the viewer also communicates with the kernel using UDP. The architecture of the simulator is illustrated in Figure 4.

#### 4.1 Burning Levels and Evaluation Points

Each building in the simulator has a *burning\_level* which indicates the state of the building. A building which is not burning has level 0. A building in the early stages of a fire has level 1. Level 2 indicates a building is in the middle stages of a fire, and level 3 indicates the later stages of a fire. When a building is burnt down, it has level 4. Levels 5

**Fig. 4.** Architecture of Simulator**Table 1.** Burning Levels and Evaluation Values

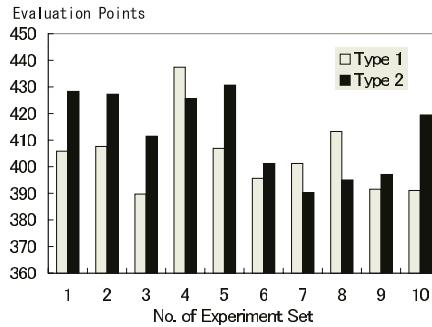
Burning level	State of building	Evaluation values
0	Not burning	6
1	The early stages of a fire	3
2	The middle stages of a fire	2
3	The later stages of a fire	1
4	Burnt down	0
5	Extinguished before burning level reached beyond 2	5
6	Extinguished on or after level 3	4

and 6 denote buildings in which the fire has been successfully extinguished. A building which is extinguished before its burning level reached beyond 2 has level 5 and if it was extinguished on or after level 3 it has level 6. To begin with, all buildings in the simulator have level 0 except for those buildings which are initially ignited. The ignited buildings start at level 1 and the burning levels increase over time to level 4. Moreover, fire spreads to buildings that have level 0 as time passes. The goal of the agents in the simulator is to extinguish the fire so that it does not spread.

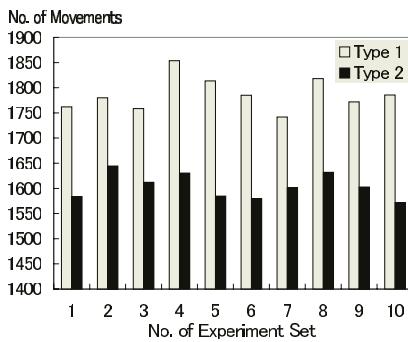
To evaluate whether or not the agents efficiently achieve their goals, we calculate an evaluation value according to the burning levels of the buildings. The relation between the burning levels and the evaluation values is shown in Table 1. We total the number of evaluation points for all of the buildings at the end of a simulation to evaluate the efficiency of the agents.

## 4.2 Estimation

Using the simulator, we evaluated the benefits of our model by comparing the following two types of agents:



**Fig. 5.** The Averages of the Total Number of Evaluation Points



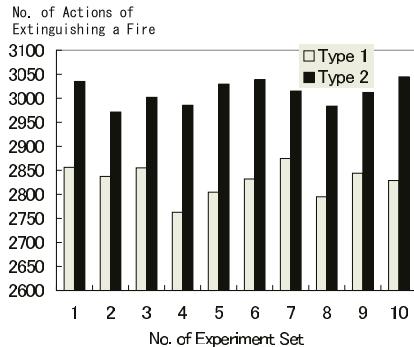
**Fig. 6.** The Average Number of Agent Movements

**Type 1** The agents form groups and cooperate with each other, but do not share actions.

**Type 2** Our model : The agents form groups, cooperate with each other and share actions.

To evaluate whether or not the agents efficiently achieve their goals, we total the evaluation points as described in Subsection 4.1 and count the total number of movements of the agents and the number of fires extinguished. We performed a total of 10 different experiments(i.e. different ways of spreading a fire), simulated each one 100 times, and calculated the averages for each experiment.

Figure 5 illustrates the averages of the total number of evaluation points for each experiment. The graph indicates that the agents in our model more efficiently extinguish fires than agents which only cooperate with each other, but do not share actions. Figure 6 illustrates the average number of agent movements in each experiment. The graph clearly shows our model reduces the number of redundant actions. Figure 7 illustrates the average number of actions of extinguishing a fire in each experiment. The graph indicates that the agents in our model perform a greater number of actions of extinguishing a fire than agents which only cooperate with each other. Table 2 gives the averages over all the experiments. The results of these simulations show that our model not only reduces redundant actions, but also increases the number of important actions.

**Fig. 7.** The Average Number of Actions of Extinguishing a Fire**Table 2.** Averages of Each Item

Type	No. of movements	No. of extinction	Points
1	1787.00	2829.01	404.03
2	1604.21	3011.76	412.64

## 5 Conclusion

In this paper, we have proposed a cooperative model for multi-agent systems which improves the performance of the systems by reducing redundant actions among the agents. We have used a simple disaster simulator to confirm that our model is efficient. The results of the simulations indicated that our model reduces the number of redundant actions(the total number of movements) and is more efficient than the model in which agents cooperate but do not share actions.

In this paper we have studied how to share actions among agents within a group. In future, we would like to study how actions could be shared among groups themselves. Furthermore, we would like to introduce machine learning techniques for determining the similarity of agents' goals and how best to distribute actions to the agents in a group.

## References

1. (Robocuprescue official web page.) In <http://www.r.cs.kobe-u.ac.jp/robocup-rescue>.
2. Ossowski, S.: Co-ordination in Artificial Agent Societies. Lecture Notes in Artificial Intelligence 1535. Springer (1999)
3. Carlos A. Iglesias, M., Jose C. Gonzalez: A survey of agent-oriented methodologies. In: Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures and Languages(ATAL98), Lecture Notes in Artificial Intelligence, Springer-Verlag, Heidelberg (1999)
4. R. Davis, R. G. Smith: Negotiation as a metaphor for distributed problem solving. In: Artificial Intelligence. Volume 20, No.1. (1983) 63–109
5. M.Tambe: Towards flexible teamwork. Journal of Artificial Intelligence Research 7, No.4 (1997)

6. Cohen, P., Levesque, H., Smith, I.: On team formation. *Contemporary Action Theory* **2** (1997) 87–114
7. Haddadi, A., ed.: *Communication and Cooperation in Agent System*. Number 1056 in LNAI, Springer (1996)
8. Nakayama, Y.: Communication and attitude change. In: *The Second International Conference on Cognitive Science and The 16th Annual Meeting of the Japanese Cognitive Science Society Joint Conference (ICCS/JCSS99)*. (1999)
9. N. Ito, T., N. Ishii: A cooperative agent model by forming a group. In: *2002 IEEE International Conference on Industrial Technology Proceedings*. Volume 2. (2002) 1260–1265

# Applying Model Checking Techniques to Game Solving

Gihwon Kwon\*

Department of Computer Science, Kyonggi University  
San 94-6, Yieu-Dong, Paldal-Gu, Suwon-Si, Kyonggi-Do, Korea  
khkwon@kyonggi.ac.kr

**Abstract.** In this paper, we automatically solve Push-Push game with model checking techniques which exhaustively explores all search space. Although model checking finds out the best solution for the game, it always suffers from the state explosion problem. To overcome this well-known problem, we use clever methods such as abstraction and pruning. In addition, we propose the relay model checking which decomposes the whole problem into a series of smaller one and then conquers one by one at a time. As a result, we solved all 50 levels of the game. Our solution shows better performance than human experts; that is, we solved the game with on average 22 fewer steps than human experts.

**Keywords:** Model checking, game solving, state explosion problem, counterexample

## 1 Introduction

Given a finite state model  $M$  and a temporal logic formula  $\phi$ , model checking determines whether the model satisfies the formula, written  $M \models \phi$  [1]. For a last decade, such a model checking technique was widely used in formal verification of hardware and software systems. Compared to theorem proving, it does not require human intervention so that verification is fully automatic. In addition, it gives a counterexample when a formula to be checked is not satisfied. Since it tells where this unsatisfaction occurs, it has been widely used in debugging a design, finding an automatic abstraction, and generating test cases [2,3,4]. Thus, automatic generation of counterexample has always been considered one of the main advantages of model checking. Efficient algorithms for generating counterexamples are widely used in practice [5,6].

This paper uses such a counterexample to solve Push-Push game which is a kind of block moving game. In this game, agent pushes one single ball at a time to one of adjacent cells if possible. In this sense, solving the game is to find out a path that makes agent move all given balls at the beginning into designated target cells. It is desirable for a path to be a minimal length; that is, the smaller, the better. In summary, the purpose of this game is to find out such a path which makes the game completed with minimal efforts.

The overall approach is as follows: Finite state model  $M$  and temporal logic formula  $AG \neg\phi$  is respectively constructed from the initial state and the goal state of the

---

\* This work was supported by Research Funds of Kyonggi University in 2001.

game, where  $\phi$  represents the goal state. According to the Computational Tree Logic (CTL) semantics [7], the formula  $AG \neg\phi$  means that there is no way to reach the goal state  $\phi$ . Since the game is well designed, there must be at least a solution path which makes to reach the goal state. Undoubtedly, the model does not satisfy the formula; that is,  $M \not\models AG \neg\phi$ . In this case, model checking outputs a counterexample to explain why  $AG \neg\phi$  is false, and that is the witness for  $EF \phi$  which means there is a path to reach the goal state  $\phi$ . In other words, we regard this counterexample as a solution for the game which shows how to reach the goal state from the initial state given at a start time.

In this way, we solve all 50 levels of the game. The most difficult problem we confronted is so called the state explosion problem [8]. Since a state space of the game is so huge, it couldn't handle it directly with model checking. To overcome this well-known problem, we use some clever methods such as abstraction and pruning so that we solve almost all levels except for the 50<sup>th</sup> level which is the most difficult one in the game. Thus we propose the relay model checking technique to reduce set of states to be considered at a time. As a result, the 50<sup>th</sup> level was solved. Compared to solutions given by human experts, our solutions show better performance; that is, we solve the game with on average 22 fewer moves than human experts.

The rest of the paper is organized as follows: Section 2 gives some basic concepts on how to check a property against a model and generate counterexample in the CTL model checker. Section 3 shows a short introduction of Push-Push game and presents how to model it by transition systems. Section 4 and Section 5 shows the game solving with naïve model checking techniques and clever model checking techniques, respectively. And evaluations and conclusions are given in Section 6 and Section 7.

## 2 Background

**Model.** A model is a 5-tuple  $\langle S, I, R, AP, L \rangle$ , where  $S$  is a finite set of states,  $I \subseteq S$  is the set of initial states,  $R \subseteq S \times S$  is the transition relation,  $AP$  is a finite set of atomic propositions, and the function  $L: S \rightarrow 2^{AP}$  assigns to each state a set of atomic propositions that are true at that state. The transition relation  $R$  is assumed to be total:  $\forall s \in S \bullet \exists s' \in S \bullet (s, s') \in R$ . That is, for every state  $s \in S$ , there exists a successor  $s' \in S$  with  $(s, s') \in R$ . A path is an infinite sequence of states in which each consecutive pair of states belongs to  $R$ . A state is reachable if it appears on some path starting from some initial state.

**Property.** Desired properties about a model can be specified in the Computation Tree Logic. The syntax of CTL formula is defined as follows:

$$\phi ::= \text{true} \mid p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid EX \phi \mid EG \phi \mid E(\phi_1 U \phi_2)$$

As usual,  $E$  is the existential path quantifier,  $X$  is the next-time operator,  $G$  is the global operator, and  $U$  is the until operator. Intuitively,  $EX \phi$  means that there is a successor state at which  $\phi$  is true,  $EG \phi$  means that  $\phi$  is always true for some path,  $E(\phi_1 U \phi_2)$  means that for some path,  $\phi_1$  remains true until  $\phi_2$  becomes eventually true.

EX, EG, and EU consist of an adequate set of CTL operators so that the other operators (EF, AX, AG, AU, AF) can be expressed in terms of these three operators. Assume a fixed model  $M$ . We write  $s_0 \models \phi$  if the CTL formula  $\phi$  is true at state  $s_0$ . The truth value of a CTL formula at state  $s_0$  is defined as follows:

$$\begin{aligned}
s_0 \models \text{true} \\
s_0 \models p &\quad \text{iff } p \in L(s_0) \\
s_0 \models \neg\phi &\quad \text{iff } s_0 \not\models \phi \\
s_0 \models \phi_1 \vee \phi_2 &\quad \text{iff } s_0 \models \phi_1 \text{ or } s_0 \models \phi_2 \\
s_0 \models \text{EX } \phi &\quad \text{iff there exists } (s_0, s_1) \in R \text{ such that } s_1 \models \phi \\
s_0 \models \text{EG } \phi &\quad \text{iff for some path } s_0, s_1, s_2, \dots, \text{ for all } i \geq 0, s_i \models \phi \\
s_0 \models \text{E}(\phi_1 \cup \phi_2) &\quad \text{iff for some path } s_0, s_1, s_2, \dots, \text{ there exists } 0 \leq j < i, \text{ for all } j, \\
&\quad s_j \models \phi_1 \text{ and } s_i \models \phi_2
\end{aligned}$$

**Model Checking.** We say that  $M$  satisfies  $\phi$ , written  $M \models \phi$ , if  $s \models \phi$  for each  $s \in I$ , i.e.,  $\phi$  is true at every initial state of  $M$ . The CTL model checking, given a model and a CTL formula, determines whether the model satisfies the formula. For any CTL formula  $\phi$ , let  $[\![\phi]\!]$  denote the set of states in which  $\phi$  is true. The model checking algorithm is then equivalent to determining whether the set of initial states is a subset of  $[\![\phi]\!]$ . For any state set  $X$ , we define  $\text{pre}_\exists(X) = \{s \in S \mid \exists_{s' \in s} \bullet (s, s') \in R \wedge s' \in X\}$  as the set of states with a successor in  $X$ . The following equations hold for any CTL formulas:

$$\begin{aligned}
[\![\text{true}]\!] &= S \\
[\![\text{false}]\!] &= \emptyset \\
[\![p]\!] &= \{s \mid p \in L(s)\} \\
[\![\neg\phi]\!] &= S \setminus [\![\phi]\!] \\
[\![\phi_1 \vee \phi_2]\!] &= [\![\phi_1]\!] \cup [\![\phi_2]\!] \\
[\![\text{EX } \phi]\!] &= \text{pre}_\exists([\![\phi]\!]) \\
[\![\text{EG } \phi]\!] &= \nu Z. ([\![\phi]\!] \cap \text{pre}_\exists(Z)) \\
[\![\text{E}(\phi_1 \cup \phi_2)]\!] &= \mu Z. ([\![\phi_2]\!] \cup ([\![\phi_1]\!] \cap \text{pre}_\exists(Z)))
\end{aligned}$$

where  $\mu$  and  $\nu$  are the least fixed-point and greatest fixed-point operators respectively. It can be shown that  $\text{pre}_\exists$  and these fixed points can be computed in time linear in the size of the model [1]. Because the formula is evaluated by computing predecessors of states, we say that the algorithm is based on backward traversals.

**Counterexample.** When the CTL formula is violated, the CTL model checker SMV(Symbolic Model Verifier, [9]) generates counterexample to show where this violation occurs. This paper only explains how to generate counterexample for the CTL operator AG (for other operators, consult [6]). It is clear that counterexample for AG  $\neg\phi$  is dual of a witness for EF  $\phi$ . For a formula EF  $\phi$ , the model checking algo-

rithm computes the least fixed point of the following monotone function  $\tau(Z) = [\![\phi]\!] \cup pre_{\exists}(Z)$  since the equation  $[\![EF\phi]\!] = \mu Z.([\![\phi]\!] \cup pre_{\exists}(Z))$  holds. The least fixed point can be computed by iterating the function  $\tau$  starting with the empty set of states. The sequence  $\tau^1(\emptyset) \subseteq \dots \subseteq \tau^n(\emptyset) \subseteq \dots$  converges to the least fixed point of  $\tau$  when  $\tau^n(\emptyset) = \tau^{n+1}(\emptyset)$ . In other words, the least fixed computation computes all sets  $\tau^i(\emptyset)$  until the sequence converges. Each such set  $S_1 = \tau^1(\emptyset)$ ,  $S_2 = \tau^2(\emptyset)$ , ...,  $S_n = \tau^n(\emptyset)$  is called a *stage* of the fixed computation. We denote the sequence of stages  $\langle S_1, S_2, \dots, S_n \rangle$  by  $stg(EF\phi)$ . It is easy to see  $S_n = [\![EF\phi]\!]$ . The idea is to use these stages to compute a witness based on forward traversals. Algorithm for generating a witness for  $EF\phi$  is given in Figure 1. It is clear that the witness for  $EF\phi$  will be given by a path descriptor. The auxiliary function  $post_{\exists}(X) = \{s_{\exists} \in S \mid \exists s \bullet s \in X \wedge (s, s_{\exists}) \in R\}$  in the algorithm returns the set of successors of  $s$ .

```

Algorithm print_witness(EF φ, s₀)
begin
    <S₁, ..., Sₙ> = stg(EF φ);
    j := 0;
    repeat
        j := j + 1;
        S := post_{\exists}(s_{j-1}) ∩ S_{n-j};
        choose sⱼ ∈ S
        until sⱼ ∈ S₁
        print <s₀, s₁, ..., sⱼ>
        print_witness(φ, sⱼ)
    end

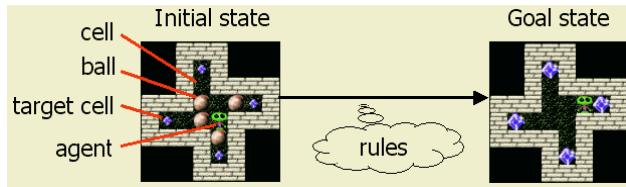
```

**Fig. 1.** Generating a witness for  $EF\phi$  (from [6]).

### 3 Modeling Push-Push Game

**Push-Push Game.** The game we consider is a kind of block-move game. In this game, agent pushes one single ball at a time to adjacent cells if possible. In this sense, solving the game is to find out a path that makes agent move all given balls at the beginning into designated target cells. It is desirable for a length of path to be a minimal one; that is, the smaller, the better. Thus, the purpose of this game is to find out such a desirable path which makes the game complete with minimal efforts. For example, Figure 2 shows the 1<sup>st</sup> level of the game which is the easiest one. In this game, agent can only push one ball at a time but you are not able to pull balls. For example, the sequence of moves  $\langle \text{down}, \text{up}, \text{left}, \text{left}, \text{right}, \text{up}, \text{up}, \text{down}, \text{right}, \text{right} \rangle$  is a solution for this game. Its length is minimal and makes the game over successfully; that is, you are able to reach the goal state (shown on the right hand side in figure 2) from the initial state (on the left hand side in the figure).

**Modeling.** How can you reach the goal state from the initial state with minimal efforts? We use the model checker SMV to answer the question. Since it takes finite

**Fig. 2.** The 1<sup>st</sup> level of Push-Push game.

state model as an input, the state space of the game should be represented as transition systems. First, we define the transition system  $T_{agent} = (S_{agent}, I_{agent}, \delta_{agent})$  for agent as follows:

- $S_{agent} = \{(x, y)\}$  is a set of locatable positions represented in an ordered pair;
- $I_{agent} \in S_{agent}$  is an initial position for an agent to be located at the beginning;
- $\delta_{agent} : S_{agent} \times D \rightarrow S_{agent}$  is the transition function; i.e.  $\delta_{agent}((x, y), d) = (x', y')$ .

In the above,  $D = \{left, right, up, down\}$  is a set of directions so that agent can select one of four directions. State transition is enabled if a move is happened and a guard condition is evaluated to be true. Thus the state transition function is defined as follows:

$$\delta_{agent}((x, y), d) = \begin{cases} (x' = x - 1, y' = y) \\ \quad \text{if } d = left \wedge movableToLeft(x, y) \\ (x' = x + 1, y' = y) \\ \quad \text{if } d = right \wedge movableToRight(x, y) \\ (x' = x, y' = y + 1) \\ \quad \text{if } d = up \wedge movableToUp(x, y) \\ (x' = x, y' = y - 1) \\ \quad \text{if } d = down \wedge movableToDown(x, y) \\ (x' = x, y' = y) \\ \quad \text{otherwise} \end{cases}$$

where  $movableToLeft(x, y)$  is a predicate which means that agent at the position  $(x, y)$  can move to its left-hand side positioned at  $(x - 1, y)$  defined as follows:

$$borderLeft(x, y) = (x - 1, y) \notin S_{agent}$$

$$borderLeftBallLeft(x, y) = (x - 2, y) \notin S_{agent} \wedge \exists_{cell} \bullet pos(cell) = (x - 1, y)$$

$$ballLeftBallLeft(x, y) = \exists_{cell_1, cell_2} \bullet (pos(cell_1) = (x - 2, y) \wedge cell_1)$$

$$\wedge (pos(cell_2) = (x - 1, y) \wedge cell_2)$$

$$movableToLeft(x, y) = \neg borderLeft(x, y) \wedge \neg borderLeftBallLeft(x, y)$$

$$\wedge \neg ballLeftBallLeft(x, y)$$

where the auxiliary function  $pos$  takes a cell and returns its position. Other predicates such as  $movableToRight$ ,  $movableToUp$ ,  $movableToDown$  can be defined in a similar way. Let  $cell_i$  be an arbitrary cell in the game. Transition system  $T_{cell_i} = (S_{cell_i}, I_{cell_i}, R_{cell_i})$  is defined as follows:

- $S_{cell_i} = B$  is a Boolean variable; i.e., 1 means it has ball and 0 means it doesn't;
- $I_{cell_i} \in S_{cell_i}$  is an initial configuration of  $cell_i$ ;
- $\delta_{cell_i} : S_{cell_i} \times D \rightarrow S_{cell_i}$  is the state transition function; i.e.  $\delta_{cell_i} : B \times D \rightarrow B$  since  $S_{cell_i} = B$ . State transition is enabled if a move is happened and a guard condition is evaluated to be true. Thus the state transition function is defined as follows:

$$\delta_{cell_i}(c, d) = \begin{cases} 0 & \text{if } c \wedge d = \text{left} \wedge \exists_{agent, cell} \bullet pos(agent) = (x+1, y) \wedge pos(cell) = (x-1, y) \wedge \neg cell \\ 1 & \text{if } \neg c \wedge d = \text{left} \wedge \exists_{agent, cell} \bullet pos(agent) = (x+2, y) \wedge pos(cell) = (x+1, y) \wedge cell \\ 0 & \text{if } c \wedge d = \text{right} \wedge \exists_{agent, cell} \bullet pos(agent) = (x-1, y) \wedge pos(cell) = (x+1, y) \wedge \neg cell \\ 1 & \text{if } \neg c \wedge d = \text{right} \wedge \exists_{agent, cell} \bullet pos(agent) = (x-2, y) \wedge pos(cell) = (x-1, y) \wedge cell \\ 0 & \text{if } c \wedge d = \text{up} \wedge \exists_{agent, cell} \bullet pos(agent) = (x, y-1) \wedge pos(cell) = (x, y+1) \wedge \neg cell \\ 1 & \text{if } \neg c \wedge d = \text{up} \wedge \exists_{agent, cell} \bullet pos(agent) = (x, y-2) \wedge pos(cell) = (x, y+1) \wedge cell \\ 0 & \text{if } c \wedge d = \text{up} \wedge \exists_{agent, cell} \bullet pos(agent) = (x, y+1) \wedge pos(cell) = (x, y-1) \wedge \neg cell \\ 1 & \text{if } \neg c \wedge d = \text{up} \wedge \exists_{agent, cell} \bullet pos(agent) = (x, y+2) \wedge pos(cell) = (x, y+1) \wedge cell \\ c & \text{otherwise} \end{cases}$$

The overall system  $T_{game} = T_{agent} \otimes T_{cell_1} \otimes \dots \otimes T_{cell_n}$  is a synchronous composition of each transition systems, where  $\otimes$  is a synchronous composition operator. Transition system  $T_{game} = (S_{game}, I_{game}, D, \delta_{game}, F_{game})$  is defined as follows:

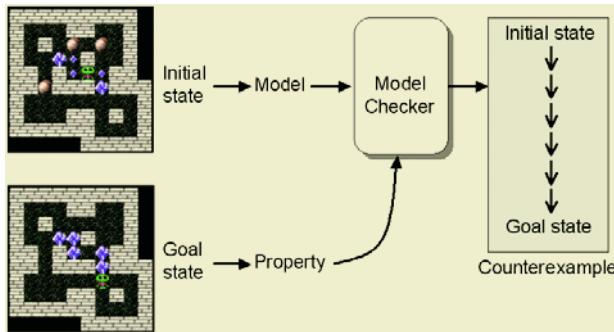
- $S_{game} = S_{agent} \times S_{cell_1} \times \dots \times S_{cell_n}$  is a set of state space the game have;
- $I_{game} = (I_{agent}, I_{cell_1}, \dots, I_{cell_n})$  is an initial state;
- $F_{game}$  is a set of target cells;
- $\delta_{game} : S_{agent} \times S_{cell_1} \times \dots \times S_{cell_n} \times D \rightarrow S_{agent} \times S_{cell_1} \times \dots \times S_{cell_n}$  is the transition function; i.e.,  $\delta_{game}((x, y), c_1, \dots, c_n, d) = (\delta_{agent}((x, y), d), \delta_{cell_1}(c_1, d), \dots, \delta_{cell_n}(c_n, d))$

**Semantics.** Semantics of the transition system is given by Kripke structure  $M = (S, I, R, X, L)$  to be constructed from  $T_{game}$  as follows:

- $S = S_{agent} \times S_{cell_1} \times \dots \times S_{cell_n} \times D$  is a set of state space;
- $I = I_{agent} \times I_{cell_1} \times \dots \times I_{cell_n} \times \{d \in D\}$  is an initial state;
- $R \subseteq S \times S$  is a transition relation such that
- $R((x, y), c_1, \dots, c_n, d, \delta_{agent}((x, y), d), \delta_{cell_1}(c_1, d), \dots, \delta_{cell_n}(c_n, d), d \in D)$
- $X = \{cell_1, \dots, cell_n\}$  is a set of atomic propositions
- $L(s) = L((x, y), c_1, \dots, c_n, d) = \{cell_1 \mid c_1\} \cup \dots \cup \{cell_n \mid c_n\}$  is labeling function.

## 4 Game Solving with Naïve Model Checking

**Approach.** Figure 3 shows our approach. The initial state and goal state in the game can be respectively translated into finite-state model and CTL property which describes there is no way to reach the goal state starting from the initial state. Obviously, it is false since there is at least one path to reach the goal state. Thus model checker generates a counterexample which shows a path to guide the goal state from the initial state. In this sense, counterexample can be regarded as a solution for this game.



**Fig. 3.** Overall approach for the game solving with model checker.

**Model in SMV.** We use the model checker SMV to solve the game. Since SMV takes finite state model as an input, the state space of the game should be represented as transition system explained in previous section. First, consider the state space of agent in the 23<sup>rd</sup> level shown in Figure 4. Agent can be located at single position at each time. Each position is represented by 2-digit integers for easy programming with SMV. The first digit denotes an x-axis position and the rest a y-axis position. In case of the 23<sup>rd</sup> level, for example, there are 39 positions so that the state space for agent is declared as follows in SMV:

```

VAR person:
{12,13,16,17,18,22,23,24,25,26,28,32,33,36,37,38,43,45,46,47,
53,55,57,61,62,63,64,65,66,67,68,71,73,76,77,78,81,82,83};

```

In addition, agent can move to one of neighbor positions in a step if possible. Thus its direction is declared as follows:

```

VAR direction: {left, right, up, down};

```

On the other hand, each cell is either empty or filled with a ball. Thus each cell is declared as a boolean variable as follows:

```

VAR
cell12:boolean; cell13:boolean;
.....
cell82:boolean; cell83:boolean;

```

Over the state space, we can define the initial states in SMV. At the beginning, agent is located at the position 55. Agent can go any direction so that a value for the variable direction can be chosen non-deterministically. Some positions such as cell24, cell47, and cell67 have a ball. Thus fragments of SMV code for representing their initial part is as follows:

```
init(person) := 55;
init(direction) := {left, right, up, down};
init(cell12) := 0;
.....
init(cell24) := 1;
.....
```

Transition relations are easily translated into SMV according to the transition system shown in the previous section.

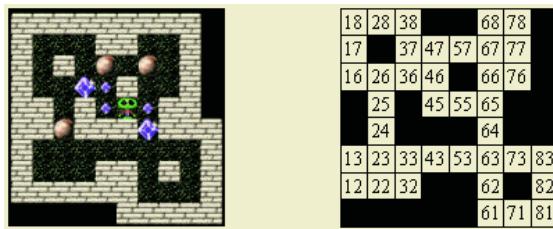


Fig. 4. State representation of the 23<sup>rd</sup> level.

**Property Specification in CTL.** The purpose of the game is to find out a minimal sequence of moves that can place all given balls into target cells. To get the solution, we have to describe the negation of the purpose; that is, there is no way to reach the goal state at which all target cell has one ball. Let  $F = \{cell_1, \dots, cell_n\}$  be the set of target cells. The corresponding property in CTL is

$$\text{AG } \neg \left( \bigwedge_{i=1}^{|F|} cell_i \in F \right)$$

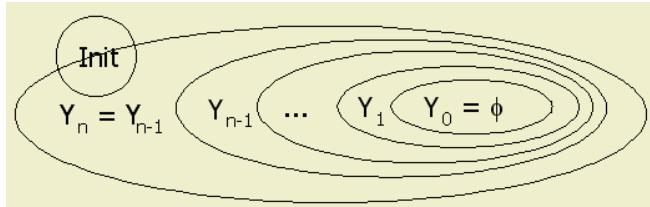
Obviously, it is false since there is at least one path to reach the goal state. Thus the model checker generates counterexample which shows how to reach the goal state from the given initial state. That is the solution for the game. In case of figure 5,  $F = \{cell36, cell45, cell46, cell64, cell65\}$ . So the property in CTL is

$$\text{AG } \neg (cell136 \wedge cell145 \wedge cell146 \wedge cell164 \wedge cell165)$$

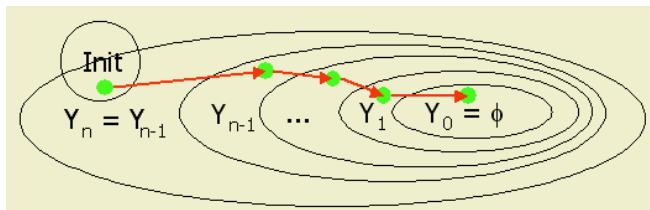
which means “there is no way to reach the goal state”.

**Game Solving with SMV.** Obviously, the above CTL is false against the finite state model since there is at least one path to reach the goal state  $\phi$ . Theoretically,  $Y_n$  denotes the set of states going to the goal state  $\phi$  in some steps; that is,  $Y_n$  is the fixed point of  $\phi$ . Thus  $Y_n \cap Init \neq \emptyset$  means that the CTL formula  $\text{AG } \neg \phi$  is false. When

the CTL formula is false, SMV generates counterexample that shows how to reach the goal state from the given initial state. That is the solution for the game.



**Fig. 5.**  $Y_n \cap \text{Init} \neq \emptyset$  means that  $\text{AG} \neg \phi$  is false.



**Fig. 6.** Generating counterexample when it is false.

**Analysis.** We performed model checking to solve the game. All the checking was carried out on a 1.99GHz Pentium 4 with 1.5GB RAM. We call it naïve model checking (NMC) and its result is given in Table 1. It took 3,000 second for the 23<sup>rd</sup> level, but the 30<sup>th</sup> level requires more than 24 hours.

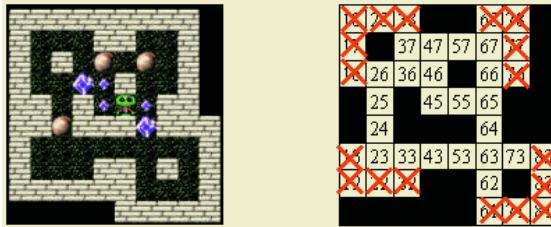
**Table 1.** Results with naïve model checking.

Level	Time(sec)	Space(MB)	BDD	Level	Time(sec)	Space(MB)	BDD
1	0.6	2	10190	26	60.0	34	1352886
2	3.6	6	162264	27	30.0	20	753887
3	33.8	32	1179417	28	420.0	210	8046131
4	0.7	3	43314	29	16.0	13	493167
5	1.0	3	45487	30	> 24 hours		
6	123.3	93	3293300	31	1020.0	500	20533835
7	53.6	37	1493151	32	> 24 hours		
8	> 24 hours			33	> 24 hours		
9	11.0	11	352558	34	> 24 hours		
10	5.0	8	260339	35	> 24 hours		
11	6.2	8	260339	36	> 24 hours		
12	10.7	12	409438	37	2200.0	854	34330495
13	355.3	240	9794593	38	28.0	18	635604
14	125.8	80	3490931	39	960.0	336	15431568
15	> 24 hours			40	> 24 hours		
16	16.5	16	514515	41	120.0	58	2245640
17	> 24 hours			42	1260.0	532	23060315
18	1200.0	508	22391859	43	480.0	299	13302604
19	1200.0	613	27678695	44	1860.0	539	23143060
20	> 24 hours			45	300.0	109	4242756
21	2880.0	1109	48330206	46	780.0	413	15742937
22	420.0	219	8571816	47	35.0	25	1019209
23	3000.0	870	34901698	48	1980.0	841	33009636
24	600.0	317	14238083	49	> 24 hours		
25	10.0	305	13229503	50	> 24 hours		

## 5 Game Solving with Clever Model Checking

As shown in Table 1, we failed to solve lots of levels with naïve model checking techniques due to a huge number of states. For solving these unsolved levels, we used clever model checking techniques such as abstraction and pruning.

**Abstraction.** To reduce the state explosion problem, abstraction is frequently used in the model checking community [10]. In this game, we found there are some positions at which balls never be placed if a gamer is wise. The model checker SMV tries to find out a minimal length of counterexample as can as possible. In this sense, model checker looks like a wise game player. Thus we removed them without violating the property. In case of the 23<sup>rd</sup> level, for example, we can remove 18 positions so that the total state space can be significantly reduced.



**Fig. 7.** Abstraction of the state space for positions; X denotes unnecessary positions at which balls never be placed if a game player is wise.

**Table 2.** Results with clever model checking.

Level	Time(sec)	Space(MB)	BDD
1	0.1	< 1	6523
2	0.5	5	53669
3	2.6	7	181424
4	0.5	< 1	51241
5	0.3	< 1	23324
6	7.5	11	399445
7	8.1	13	361845
8	53.9	105	4037813
9	0.3	< 1	23287
10	0.4	< 1	34465
11	0.3	< 1	32802
12	1.5	6	135401
13	0.4	< 1	29385
14	2.3	7	172012
15	167.8	190	8495225
16	0.5	< 1	45514
17	415.0	470	13184027
18	8.1	21	586773
19	209.0	325	11037872
20	224.0	358	13007571
21	14.8	26	893813
22	52.5	59	2337064
23	79.0	67	4278917
24	7.1	17	611142
25	13.5	25	697371

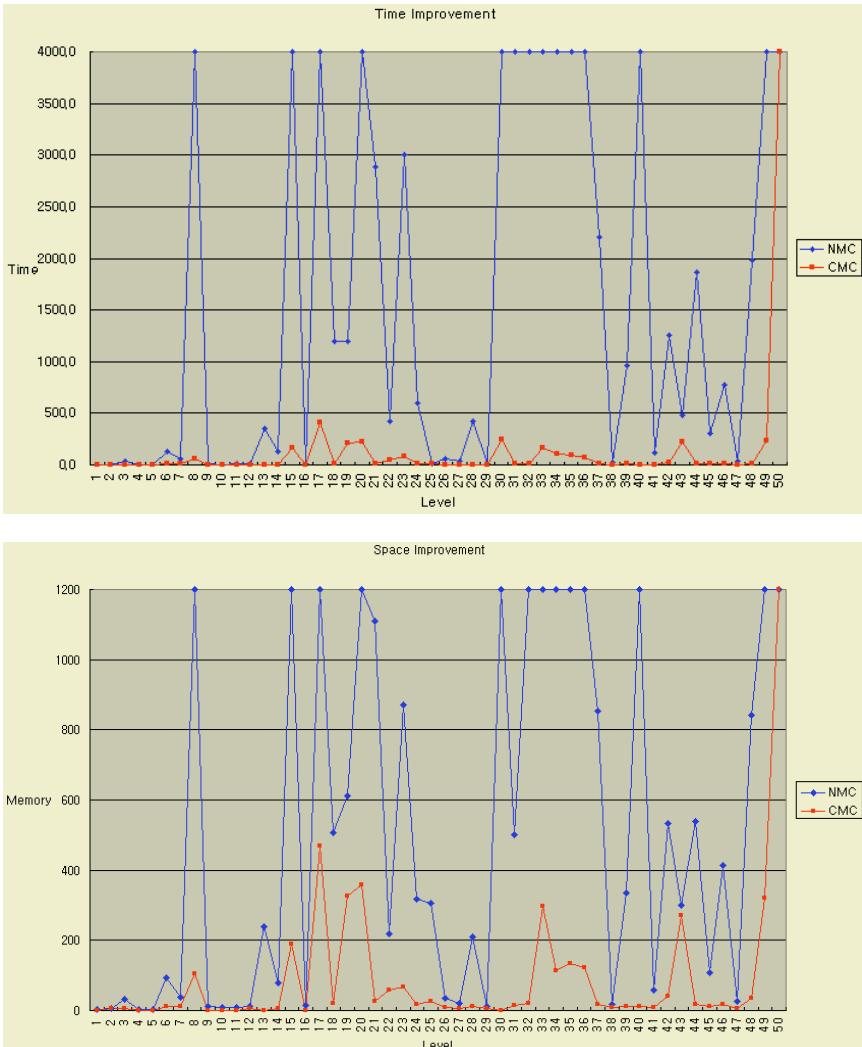
  

Level	Time(sec)	Space(MB)	BDD
26	5.4	8	269886
27	3.0	7	233811
28	4.8	13	376884
29	1.4	5	118553
30	311.1	796	32522648
31	8.3	16	655700
32	10.1	20	767277
33	168.2	298	9446378
34	101.8	113	4510854
35	98.6	133	5571438
36	73.6	122	5004938
37	8.0	18	608666
38	1.9	8	203503
39	10.4	11	399459
40	4.3	13	357824
41	3.4	10	310274
42	25.0	40	1195649
43	223.3	272	11769524
44	12.9	18	659379
45	6.5	13	336695
46	10.2	17	595338
47	1.8	5	134084
48	16.6	35	1390943
49	234.2	321	10880846
50	> 24 hours		

**Pruning.** We can greatly simplify the search of the state space by observing that person and balls never be placed at the same position. This invariant can be used to prune unnecessary search space [11]. This can be done in SMV by putting the following invariant in an INVAR statement as follows:

```
INVAR !(cell123&agent=23)
INVAR !(cell124&agent=24)
...
```

**Analysis.** With the help of clever methods, we solved almost all levels shown in Table 2. Compared to the result NMC, we had great improvements in time and space with clever model checking (CMC) in figure 8.



**Fig. 8.** Time and space improvements with clever model checking techniques.

**Relay Model Checking.** Although almost all levels were solved with clever methods mentioned just above, the last 50<sup>th</sup> level was not solved due to huge state space. In general, the divide-and-conquer principle is widely used in all areas of computer science. We use this principle to solve it. The full CTL property for this level is

$$\text{AG } \neg(\text{cell41} \wedge \text{cell42} \wedge \text{cell51} \wedge \text{cell52} \wedge \text{cell61} \wedge \text{cell62} \wedge \text{cell71} \wedge \text{cell72} \wedge \text{cell81} \wedge \text{cell82})$$

We split the formula into the four parts as below:

$$\begin{aligned} \text{AG } \neg(\text{cell42}) \\ \text{AG } \neg(\text{cell52} \wedge \text{cell62}) \\ \text{AG } \neg(\text{cell72} \wedge \text{cell41} \wedge \text{cell51}) \\ \text{AG } \neg(\text{cell61} \wedge \text{cell71} \wedge \text{cell81} \wedge \text{cell82}) \end{aligned}$$

In the first round of model checking, only the formula AG  $\neg(\text{cell42})$  is checked against the original model. Since it is false, SMV generates a counterexample which shows how agent pushes a ball into the cell42 which is placed at the most inner side. We get another model by modifying the original model with counterexample generated in the first round which is input the next round. In the second round of model checking, AG  $\neg(\text{p52} \wedge \text{p62})$  is checked against the modified model. In this way, we solved the 50<sup>th</sup> level in four rounds shown in Figure 9 and its result is shown in Table 3. Since it looks like a relay game, we call it *relay model checking*.



Fig. 9. Relay model checking for the 50<sup>th</sup> level.

Table 3. Results with relay model checking.

Level	Round	Time(sec)	Memory(MB)	BDD
50	1	420.0	460	15962368
	2	600.0	764	29110962
	3	360.0	454	18660960
	4	540.0	762	29046274

## 6 Evaluations

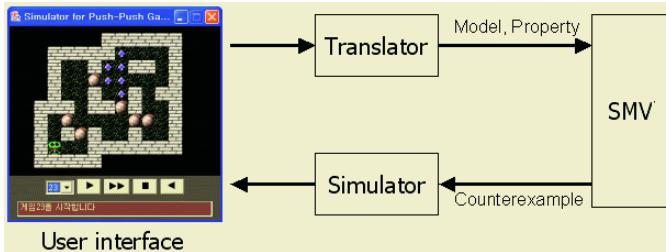
**Efficiency.** Push-Push game is so popular that its solutions are opened. We compared their answers with our ones. As a result, our solutions are far more efficient than their solutions. They found the answer manually, but we found it mechanically with model checker. For level 23, experts have 219 moves to complete it, but we have only 122 moves; that is, we solved it with 97 fewer moves. On average, we take 22 less moves than experts shown in Table 4.

**Implementation.** The schematic view of the implementation is shown in figure 10. Translator automatically translates each level into the input language SMV according

**Table 4.** Model checker vs. Human experts.

(Three levels 45, 46 and 50 are not comparable since experts do not give a complete sequence of moves)

Level	Experts	MC	Difference	Efficiency	Level	Experts	MC	Difference	Efficiency
1	10	10	0	0.0%	26	106	83	23	21.7%
2	93	89	4	4.3%	27	65	61	4	6.2%
3	120	114	6	5.0%	28	209	162	47	22.5%
4	36	33	3	8.3%	29	95	89	6	6.3%
5	50	50	0	0.0%	30	124	101	23	18.5%
6	94	79	15	16.0%	31	194	116	78	40.2%
7	45	44	1	2.2%	32	160	126	34	21.3%
8	213	170	43	20.2%	33	301	261	40	13.3%
9	38	34	4	10.5%	34	89	59	30	33.7%
10	63	29	34	54.0%	35	100	56	44	44.0%
11	47	29	18	38.3%	36	191	154	37	19.4%
12	60	56	4	6.7%	37	165	120	45	27.3%
13	55	55	0	0.0%	38	58	38	20	34.5%
14	86	72	14	16.3%	39	88	64	24	27.3%
15	115	101	14	12.2%	40	83	77	6	7.2%
16	66	64	2	3.0%	41	114	108	6	5.3%
17	119	97	22	18.5%	42	151	137	14	9.3%
18	141	129	12	8.5%	43	67	35	32	47.8%
19	76	60	16	21.1%	44	203	171	32	15.8%
20	130	105	25	19.2%	45	122	169	- 47	N/A
21	67	57	10	14.9%	46	72	106	- 34	N/A
22	88	82	6	6.8%	47	101	96	5	5.0%
23	219	122	97	44.3%	48	230	173	57	24.8%
24	115	89	26	22.6%	49	136	84	52	38.2%
25	99	83	16	16.2%	50	78	341	- 263	N/A

**Fig. 10.** Implementation.

to rules given in Section 3. Since each game has a solution, SMV generates a counterexample which is a solution for the game. Simulator takes counterexample generated from SMV, parses them, and moves agent according to the generated counterexamples. The implementation is available on our website and all solutions as well.

## 7 Conclusions

Model checking exhaustively explores all search space generated by the game to find out a minimal sequence of moves. In the Push-Push game, the search space often is huge. Although model checking finds out the exact solution, it always suffers from the state explosion problem. To overcome this well-known problem, we use some

clever methods such as abstraction and pruning so that we manage to solve the game up to the 49<sup>th</sup> level. However, we could not solve the 50<sup>th</sup> level with these methods because its search space is extremely huge. Thus we proposed the relay model checking technique to reduce set of states to be considered at a time. Finally, the last level was solved with relay model checking.

In summary, we solved all 50 levels of the game with model checking techniques. Our solutions show better efficiency than human experts; that is, less moves is required to complete the game. We believe that our experiences help a game designer to devise a more complicated game and have a confidence that there is at least one possible solution in their game at a design stage. It makes the game development time short and makes the game quality high. As future works, we will apply these experiences to other games.

## References

1. E.M. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 1999.
2. R. Hojati, R.K. Brayton, and R.P. Kurshan, "BDD-based Debugging of Designs using Language Containment and Fair CTL," in *Proceedings of Computer Aided Verification*, 1993.
3. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith., "Counterexample-Guided Abstraction Refinement," in *Proceedings of Computer Aided Verification*, pp.154-169, 2000.
4. P.E. Ammann, P.E. Black, and W. Majurski, "Using Model Checking to Generate Tests from Specifications," in *Proceedings of ICFEM '98*, pp.46-54, 1998.
5. E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao, "Efficient Generation of Counterexamples and Witness in Symbolic Model Checking," in *Proceedings of Design Automation Conference*, pp.427-432, 1995.
6. E.M. Clarke, S. Jha, Y. Lu, and H. Veith, "Tree-like Counterexamples in Model Checking," in *Proceedings of Logic In Computer Science*, 2002.
7. E.A. Emerson. Temporal and modal logic. in the *Handbook of Theoretical Computer Science: Formal Models and Semantics*, J. van Leeuwen, editor, pp. 995-1072, Elsevier, 1990.
8. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Progress on the State Explosion Problem in Model Checking," in *Proceedings of 10 Years Dagstuhl, LNCS 2000*, pp.154-169, 2000.
9. K.L. McMillan, *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.
10. Y. Lu, *Automatic Abstraction in Model Checking*, Ph.D. thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering, 2000.
11. W. Chan, *Symbolic Model Checking for Large Software Specifications*, Ph.D. thesis, University of Washington, Computer Science and Engineering, 1999.

# A Fuzzy Logic-Based Location Management Method for Mobile Networks

Ihn-Han Bae<sup>1</sup>, Sun-Jin Oh<sup>2</sup>, and Stephan Olariu<sup>3</sup>

<sup>1</sup> School of Computer and Information Comm. Eng., Catholic University of Daegu  
Gyeongbuk 712-702, South Korea  
[ihbae@cu.ac.kr](mailto:ihbae@cu.ac.kr)

<sup>2</sup> Department of Computer and Information Science, Semyung University  
Chungbuk 390-711, South Korea  
[sjoh@telcom.semyung.ac.kr](mailto:sjoh@telcom.semyung.ac.kr)

<sup>3</sup> Department of Computer Science, Old Dominion University  
Norfolk VA 23529-0162, USA  
[olariu@cs.odu.edu](mailto:olariu@cs.odu.edu)

**Abstract.** State-of-the-art wireless communication networks allow dynamic relocation of mobile terminals. A location management mechanism is required to keep track of a mobile terminal for delivering incoming calls. In this paper, we propose a fuzzy logic-based location management method to reduce paging cost. In the proposed method, the location update uses an area-based method that uses direction-based method together with movement-based method, and the location search uses a fuzzy logic-based selective paging method based on the mobility information of mobile terminals. A partial candidate paging area is selected by fuzzy control rules, and then the fuzzy logic-based selective paging method pages only cells within the partial candidate paging area. The performance of the proposed fuzzy logic-based location management method is evaluated by an analytical model and by a simulation, and is compared with that of basic velocity paging (BVP) method.

## 1 Introduction

In recent years, the number of subscribers for the Personal Communication Services (PCS) has increased rapidly. The most popular solution to support the growing population of mobile users is to reduce the cell size to promote bandwidth reuse. However, a serious problem resulting from using small cells is the high cost of location management. There are two basic operations in location management: location update and paging. The operation of recording the current location of a mobile terminal (MT) is called location update while the operation of searching the exact location of an MT is called paging [1].

The location management system cannot keep track of MTs without using a location updating mechanism, called registration process. The registration process consumes a significant amount of the MT battery power, air-interface bandwidth, wireless network bandwidth, system computing resource and other system resources. It is desirable to reduce the number of registrations to its minimal. Unfortunately, reducing

the number of registrations decreases the accuracy of the current location information for the high mobility MTs. For a high mobility MT at the standby state, the last update location may not be the current location of the mobile host. Therefore, when an incoming call arrives, the network must deliver the signal to the MT via a paging process. Since the location information may not be accurate, the paging process finds out the current location by polling the cells close to the last updated location. For the existing wireless networks, the paging process also consumes significant amount of system resources. In order to reduce the total location management cost, it is necessary to strike a balance between the paging process and registration process [2], [3].

The main contribution of this paper is to propose a fuzzy logic-based location management method on the basis of the mobility information of MTs to reduce paging cost by estimating the paging area. In the proposed method, location update uses the area-based method that uses movement-based method together with direction-based method, and location search uses the fuzzy logic-based selective paging method that is based on the basic fuzzy sets for the movement direction and the movement level of the MT.

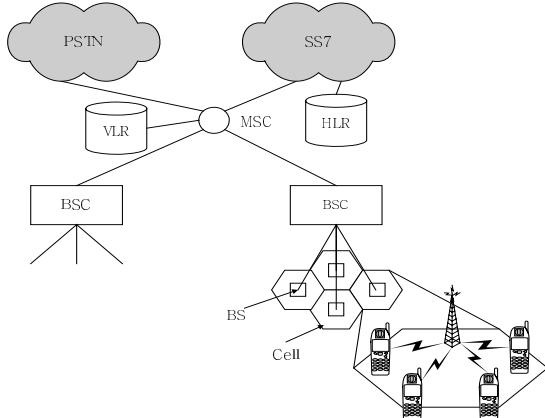
When an incoming call arrives, the system computes the basic fuzzy set for the movement level of the MT according to current time, predicts the partial candidate paging area that the MT may be reside by the fuzzy logic based on the basic fuzzy sets for the movement direction and the movement level of the MT, and pages only cells within the partial candidate paging area. We evaluate the performance of the proposed method by both an analytical model and a simulation, and the performance is compared with that of Wan's basic velocity paging method [2], [3].

This paper is organized as following. Sect. 2 presents a broad PCS architecture. Sect. 3 summarizes related works. Sect. 4 describes the proposed fuzzy logic-based location management method. Sect. 5 evaluates the performance of the proposed method. Finally, Sect. 6 offers concluding remarks.

## 2 The PCS Architecture

Wireless bandwidth is inherently a scarce resource, and the number of wireless channels available for mobile communications is very limited. In order to support a large number of PCS subscribers with a relatively small number of wireless channels, most current PCS networks adopt a cellular architecture. Under this scheme, the service coverage area is divided into cells, and each cell is assigned a number of wireless channels. The number of active connections within each cell cannot exceed the number of channels available. There is a base station (BS) installed in each cell, and MTs within a cell communicate with the BS through wireless links.

Fig. 1 demonstrates the general architecture of a PCS network. Several BSs are connected to a base station controller (BSC). The primary function of the BSC is to manage the radio resources of its BSs, such as by performing handoff and allocating radio channels. The BSC is connected to a mobile switching center (MSC), which provides typical switching functions and coordinates location registration and call delivery.



**Fig. 1.** PCS network architecture.

The MSC is connected to both the backbone wireline network such as the public switched telephone network (PSTN) and the signaling network such as the signaling system no. 7 (SS7). Current schemes for location management are based on a two level data hierarchy such that two types of database, the home location register (HLR) and the visitor location register (VLR), are involved in tracking an MT. In general, there is an HLR for each PCS network, and a user is permanently associated with an HLR in his/her subscribed PCS network. Information about each user, such as the types of services subscribed, billing information, and location information, are stored in a user profile located at the HLR. The number of VLRs and their placement vary among networks. Each VLR stores the information of the MTs (downloaded from the HLR) visiting its associated area [4].

Current PCS systems adopt an approach such that the network coverage area is partitioned into registration areas (RAs), and the MT performs a location update whenever it enters a new RA. Each RA consists of a number of cells; and in general, all base stations belong to the same RA are connected to the same MSC.

### 3 Related Works

Location management includes two major tasks: location registration and call delivery. Location registration procedures update the location databases (HLR and VLRs) and authenticate the MT when up-to-date location information of an MT is available. On the other hand, call delivery or paging involves the querying of location databases to determine the current location of a called MT. Paging process relies on the location information provided by registration process. The cost of paging also depends on the cooperating registration scheme. Generally, the more accurate location information the registration provides, the less cells the system needs to page. However, accurate location information does incur higher overhead in the form of registration cost. Therefore, a balance between the paging and registration schemes is necessary to achieve cost reduction.

Existing location update methods are as follows [2], [3], [4]:

**Location Area Registration:** In this method, the collection of all cells in the system is partitioned into a number of disjoint location areas. Each cell in a location area broadcasts the location area ID to tell mobile terminals which location area they reside in. A mobile terminal registers whenever it crosses the boundary of two location areas.

**Time Based Registration:** A mobile terminal registers whenever its timer has expired. The timer is set according to a system parameter and reset each time after expiration.

**Distance Based Registration:** A mobile terminal registers itself whenever the distance between the current cell and the last registered cell exceeds a threshold.

**Movement Based Registration:** A mobile terminal registers whenever the number of cells it traveled since the last registration exceeds a threshold that is called movement threshold.

**Direction Based Registration:** It is assumed that the movement of a mobile terminal can be divided into steps and each step has a destination. The MT inspects its direction periodically and a location update will be generated when its direction change is greater than the direction threshold defined for the step [1].

**State Based Registration:** A mobile terminal registers when the expected cumulative per-unit-time cost of paging and registration is minimized. The expected cost is based on the mobile unit states, which include the location, the time elapsed, and functions of the past history.

**Power-Up and Power-Down Registration:** A mobile terminal registers whenever it powers up or powers down.

The paging methods cooperated with location update methods are as follows [2], [3]:

**System Wide Paging:** Simply pages all the cells in the system looking for a particular mobile terminal. It is the simplest paging method of all, but also the most expensive one. This method has been generally abandoned in current cellular/PCS systems because of the high cost.

**Location Area Paging:** Pages all the cells in the mobile's current location area. This method requires the location area registration method to cooperate with it. This scheme has been adapted by several cellular phone network implementation for its acceptable delay at field test. However, this approach incurs much higher paging overhead in comparison with selective paging scheme.

**Selective Paging:** The paging process consists of iterative steps. In each step, a subset of the cells is selected for paging according to predetermined selecting criteria (e.g. distance). The paging process terminates as soon as the mobile terminal is found. The paging delay is the major problem with this scheme.

G. Wan and E. Lin proposed a dynamic paging algorithm based on the semi-realtime mobile terminal movement information to predict the paging area thus, to reduce the paging cost [2], [3]. The dynamic paging algorithm, namely velocity-paging scheme cooperates with a movement-based registration, and the concept of velocity class was introduced to characterize the velocity of an MT. A velocity class is defined as a range of velocity. For the movement based registration, the number of moves made during a predetermined time period can define a velocity class, called the

velocity time unit (VTU). For example, if the VTU is 10 minutes, the movement threshold,  $k$ , is 3, and a mobile terminal registers in 15 minutes,  $t$ , it has a velocity class index (VCI) of 2.

$$VCI = \frac{k}{t} \times VTU = \frac{3}{15} \times 10 = 2 \quad (1)$$

The system calculates the maximum distance the MT could have traveled since the last registration. A candidate paging area should be located within this distance from the last known location of the MT. The candidate cells of the paging area are the circular area centering on the last known location of the MT with the radius of this distance. Therefore, the BVP scheme pages only these candidate cells.

## 4 Fuzzy Logic-Based Location Management Method

In this Sect., the fuzzy logic-based location management method is presented. In our method, we propose an area-based registration method that uses movement-based method together with direction-based method, and design a fuzzy logic-based selective paging method that cooperates with the area-based registration. In the proposed method, the system obtains the mobility characteristic of each MT for the location registration and paging. It performs the location registration process based on the mobility information of the MT, gets basic fuzzy sets for movement direction and movement level of the MT, selects a partial candidate paging area by using the fuzzy logic rule on the basis of the basic fuzzy sets. Therefore, the system pages only the cells within the selected partial candidate paging area.

### 4.1 Area-Based Registration

An area-based location registration process needs the mobility information for movement level and movement direction of the MT because it uses movement-based and direction-based registrations. To get the information for movement level of an MT, the MT keeps a counter of moves and increments it every time it moves to another cell. We assume that the system knows current movement direction of the MT. If the counter exceeds the movement threshold  $k$ , the MT registers its current location to the serving VLR. The system calculates the estimated movement direction of the MT on the basis of the current observed movement direction in the current registration and the estimated movement direction in the last registration. To estimate the movement direction of the MT, we use the error-based filter of exponentially weighted moving average (EWMA) [7] such as:

$$\theta_t = \alpha_t \theta_{t-1} + (1 - \alpha_t) \Theta_t \quad (2)$$

where  $\theta_t$  is the newly estimated movement direction,  $\theta_{t-1}$  is the prior estimated movement direction, and  $\Theta$  is the current observed movement. The term  $\alpha_t$  is called *gain*, and determines the filter's reactivity. The error at each individual observation is the difference between the past estimate and the current observation:  $|\theta_{t-1} - \Theta_t|$ . Rather than

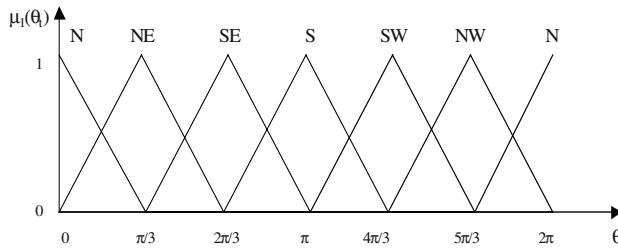
use raw error values at each step, these errors are filtered through a secondary EWMA filter. Estimator error,  $\Delta_t$  is:

$$\Delta_t = \gamma \Delta_{t-1} + (1 - \gamma) |\theta_{t-1} - \Theta_t| \quad (3)$$

where  $\gamma$  is 0.6, We then set the gain of the Error-based filter to be:

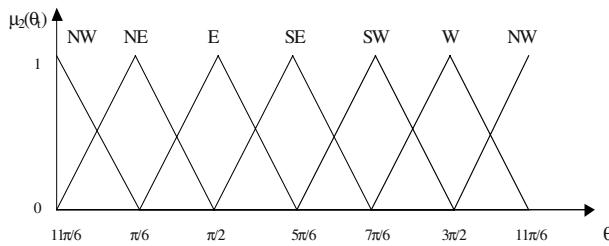
$$\alpha_t = 1 - \frac{\Delta_t}{\Delta_{\max}} \quad (4)$$

where  $\Delta_{\max}$  is the largest instability seen in the ten most recent observations.



**Fig. 2.** The first membership function  $\mu_1(\theta)$  and basic fuzzy sets for movement direction.

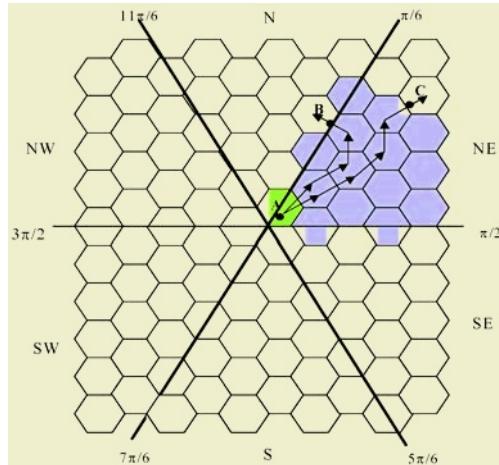
From Eq. (2), the estimated movement direction  $\theta_t$  is calculated by using the magnetic north angle that turns to clockwise from the north ( $0^\circ$ ). To reduce frequently occurred registration updates from zigzag moves of MTs in the boundary area with neighbor areas, we use two membership functions to select a candidate paging area for estimating movement direction. The estimated movement direction is mapped to a basic fuzzy set by the membership function that has larger membership value between two membership functions as shown in Fig. 2 and Fig. 3.



**Fig. 3.** The second membership function  $\mu_2(\theta)$  and basic fuzzy sets for movement direction.

For the example of Fig. 4, if we assume that the last registration of an MT was occurred at cell A, a movement threshold is 4, and an estimated movement direction is  $\pi/3$ . For the estimated direction, the membership value of the first membership function is  $\mu_1(\pi/3)=1.0$ . On the other hand, the membership value of the second membership function is  $\mu_2(\pi/3)=0.5$ . Therefore, the basic fuzzy set for the MT's direction is decided by the first membership function, thus the system predicts that the MT moves

to NE. While the MT moves cells within movement threshold and to NE direction, location registration does not occur. If the MT moves from cell A to cell C that exceeds movement threshold or if the MT moves from cell A to cell B that changes movement direction from NE to N, the location registration is occurred.



**Fig. 4.** Example of area-based registration.

To effectively support the area-based registration, MTs keep the mobility information as shown in Fig. 5, where MT\_ID is the MT identifier, Move\_Counter is the number of cells that the MT moves after the last registration, and Candidate\_Area\_Cells is the set of cells that is not only within movement threshold but also to the same basic fuzzy set for movement direction, the set of cells within candidate paging area that the MT may reside in the near future.

MT_ID	Move_Counter	Candidate_Area_Cells
-------	--------------	----------------------

**Fig. 5.** Mobility information.

Each MT increments the Move\_Counter every time it moves to another cell. The BS of each cell checks the Move\_Counter value of entering MT. If this value exceeds the movement threshold or the cell ID doesn't exist in Candidate\_Area\_Cells, performs location registration process. Then, the Move\_Counter is initialized to 0, and Candidate\_Area\_Cells is reconstructed according to the estimated movement direction. If the registration is occurred to the movement direction, the current movement direction is set to the estimated movement direction, and the estimator error array that is stored in VLR is initialized to 0.

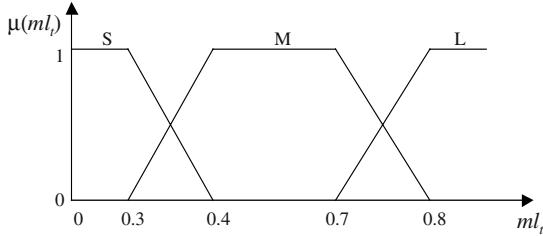
#### 4.2 Fuzzy Logic-Based Selective Paging

We can predict the movement level of an MT using Wan's VCI [2, 3]. If the system knows movement threshold, cell radius, the last registration time of the MT and cur-

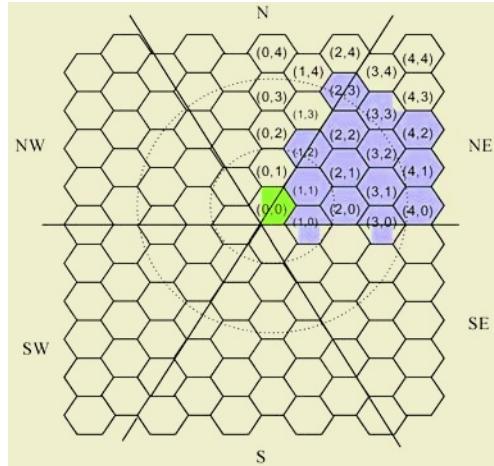
rent time, the system can estimate current velocity of the MT. When an MT is registered, the estimated velocity of the MT can be computed by EWMA that is discussed in Sect. 4.1. Thus, the time ( $T$ ) that the MT moves cells as much as the movement threshold ( $k$ ) can be calculated, and the current movement level of the MT at current time  $t$ ,  $ml_t$  is calculated by Eq. (5).

$$ml_t = \frac{t}{T} \quad (5)$$

We map the calculated  $ml_t$  to the basic fuzzy set for movement level by using the membership function as shown in Fig. 6.



**Fig. 6.** The membership function and basic fuzzy sets for movement level.



**Fig. 7.** Candidate paging area and cell coordinates.

From previous Sect., we know that the system can select a candidate paging area that the MT may reside in the near future. In case that the movement threshold is 4 and the estimated direction of an MT by the first membership function is NE, the candidate paging area and cells coordinated within the area are shown in Fig. 7.

Fig. 8 shows the fuzzy control rules for the fuzzy logic-based selective paging as shown in Fig. 7. For example, If the basic fuzzy set for movement direction is NE and the basic fuzzy set for movement level is S, the system selects a partial candidate paging area  $PCA_{NES}$ , pages only (0, 0), (1, 0) and (1, 1) cells. Also, the system has

other fuzzy control rules for selective paging according to the second membership function for direction  $\mu_2(\theta)$ . The other fuzzy control rules are 18 similar to Fig. 8. Accordingly, the total number of fuzzy control rules for the fuzzy logic-based location management is 36.

DIR \ MOV	S	M	L
N	PCA <sub>NS</sub>	PCA <sub>NM</sub>	PCA <sub>NL</sub>
NE	PCA <sub>NES</sub>	PCA <sub>NEM</sub>	PCA <sub>NEL</sub>
SE	PCA <sub>SES</sub>	PCA <sub>SEM</sub>	PCA <sub>SEL</sub>
S	PCA <sub>SS</sub>	PCA <sub>SM</sub>	PCA <sub>SL</sub>
SW	PCA <sub>SWS</sub>	PCA <sub>SWM</sub>	PCA <sub>SWL</sub>
NW	PCA <sub>NWS</sub>	PCA <sub>NWM</sub>	PCA <sub>NWL</sub>

(Input variables)  
MOV: the fuzzy sets for movement level  
S - Short, M - Middle, L – Long  
DIR: the fuzzy sets of movement direction  
N – North, NE – North East, SE – South East  
S – South, SW – South West, NW – North West  
(Output variables) the cells within PCAs (partial candidate areas)  
PCA<sub>NES</sub>={(0,0), (1,0), (1,1)}  
PCA<sub>NEM</sub>={(2,0), (2,1), (1,2), (3,0), (3,1), (3,2), (2,2)}  
PCA<sub>NEL</sub>={(4,0), (4,1), (4,2), (3,3), (2,3)}...

**Fig. 8.** Fuzzy control rules for selective paging.

When the system initiates a call delivery for a ready MT, the fuzzy logic-based selective paging is called. The following additional steps should be performed in the fuzzy logic-based selective paging procedure:

1. The system calculates the basic fuzzy set for movement level of the searching MT.
2. Get the basic fuzzy set for movement direction of the MT that is stored in VLR.
3. Select a partial candidate paging area by using fuzzy control rules.
4. Page only cells within the partial candidate paging area.
5. If the MT is in the cells, setup a call.
6. Otherwise, pages all cells within the candidate paging area with the exception of cells within the partial candidate paging area, and setup a call.

To cooperate with the fuzzy logic-based selective paging, the following attributes are needed in the mobile's VLR record:

- the last registration location
- the basic fuzzy set for movement direction
- the last registration time
- the estimated movement direction
- the estimated velocity
- the cells within the candidate area
- the cells within the partial candidate area

## 5 Performance Evaluation

The performance of the proposed fuzzy logic-based location management is to be evaluated by both an analytical model and a simulation, where evaluation criteria is the total cost that are sum of update and query costs. The performance of our method is compared with that of BVP method.

### 5.1 Analytical Model

In the analytical model, we evaluate the performance of the proposed method in terms of the total cost that consists of registration and paging costs. The total paging cost of our method is:

$$C_{total\_paging} = S \left[ P_{hit}^{p\_area} N_{p\_area}^{cell} + (1 - P_{hit}^{p\_area}) N_{area}^{cell} \right] C_{page} \quad (6)$$

where,  $S$  is the total number of calls attempted per hour in the system,  $P_{hit}^{p\_area}$  is the probability that the locating MT is in cells of partial candidate paging area,  $N_{p\_area}^{cell}$  is the average number of cells within partial candidate paging area that is paged for call delivery,  $N_{area}^{cell}$  is the number of cells within candidate paging area, and  $C_{page}$  is the unit for paging. In the fuzzy logic-based paging,  $N_{p\_area}^{cell}$  is computed by:

$$N_{p\_area}^{cell} = P_s N_s^{cell} + P_m N_m^{cell} + P_l N_l^{cell} \quad (7)$$

where,  $P_s$ ,  $P_m$  and  $P_l$  represent the probabilities that the mobility level of an MT is S, M, and L, respectively,  $N_s^{cell}$ ,  $N_m^{cell}$  and  $N_l^{cell}$  represent the number of cells those are paged in case that the mobility level of the MT is S, M and L, respectively. From Fig. 6, we can get  $N_s^{cell}$ ,  $N_m^{cell}$  and  $N_l^{cell}$  as follows:

$$\begin{aligned} N_l^{cell} &= \sum_{i=1}^{l+1} i - \sum_{i=1}^{m+1} i = \frac{l^2 + 3l - m^2 - 3m}{2} \\ N_m^{cell} &= \sum_{i=1}^{m+1} i - \sum_{i=1}^{s+1} i = \frac{m^2 + 3m - s^2 - 3s}{2} \\ N_s^{cell} &= \sum_{i=1}^{s+1} i = \frac{s^2 + 3s + 2}{2} \end{aligned}$$

where  $s$ ,  $m$  and  $l$  are the number of moves those mobility level is S, M, and L for movement threshold  $k$ , respectively. If  $P_s$ ,  $P_m$  and  $P_l$  have the same probability,  $N_{p\_area}^{cell}$  is simplified such as:

$$N_{p\_area}^{cell} = \frac{l^2 + 3l + 2}{6} \quad (8)$$

$N_{area}^{cell}$  is computed by Eq. (9).

$$N_{area}^{cell} = \sum_{i=1}^{k+1} i = \frac{(k+1)(k+2)}{2} \quad (9)$$

The total registration cost of the fuzzy logic-based location management method is as follows:

$$C_{total\_reg} = N_{sub} \left\{ \frac{E[X]}{D} (1 + P_{update}^{dir}) \right\} C_{reg} \quad (10)$$

where,  $N_{sub}$  is the number of subscribers in the system,  $E[X]$  is the average number of cells traveled per hour,  $D$  is the movement threshold,  $P_{update}^{dir}$  is the probability that the direction-based registration occurs between the last movement-based and the current movement-based registrations, and  $C_{reg}$  is the unit cost for registration. The  $E[X]$  is computed by the following equation [2], [3], [8]:

$$E[X] = \left\lceil \frac{\sqrt{12 \frac{V^2 + \sigma^2}{\gamma^2} - 3} - 3}{6} \right\rceil$$

where,  $V$  and  $\sigma$  represent the mean velocity and the deviation of standard normal distribution for an MT, respectively, and  $\gamma$  represents the radius of a cell. Therefore, the total cost of the fuzzy logic-based location management method is as follows:

$$C_{total\_cost} = S \left\{ P_{hit}^{p-area} N_{p-area}^{cell} + (1 - P_{hit}^{p-area}) N_{area}^{cell} \right\} C_{page} + N_{sub} \left\{ \frac{E[X]}{D} (1 + P_{update}^{dir}) \right\} C_{reg} \quad (11)$$

Table 1 shows the parameter values for evaluation of an analytical model. The number of calls attempted per hour affects the performance of the location management method, so we consider two types: that is, the total number of calls attempted per hour is many ( $S=50,000$ ); the total number of calls attempted per hour is few ( $S=10,000$ ). The  $P_s$ ,  $P_m$  and  $P_l$  are determined by the  $ml_t$  value of the basic fuzzy set for mobility level from Fig. 5, and the  $P_{hit}^{p-area}$ ,  $P_{hit}^{area}$  and  $P_{update}^{dir}$  are set on the basis of the simulation results from Sect. 5.2, respectively. In the BVP method, the  $P_{hit}^{area}$  represents the probability that searching MT is in the candidate cells those are in the circular area centering on the mobile's last known location and the radius is VCI.

Fig. 9 shows the total location management cost over movement threshold in the case that the number of calls attempted in the system is many. From the result, the performance of the proposed fuzzy logic-based location management method (*Fuzzy*) is better than that of BVP regardless of movement thresholds. Specially, we know that the bigger movement threshold is, the larger performance difference between *Fuzzy* and *BVP* is.

**Table 1.** Parameter values for the evaluation of analytical model.

Parameter	Value	Comment
$S$	50000, 10000	Calls attempted in the system
$N_{sub}$	100000	Number of subscribers
$N(V, \sigma^2)$	$N(60, 5)$	Normal standard deviation for velocity
$\gamma$	1 Km	Cell radius
$P_s, P_m, P_l$	0.35, 0.4, 0.25	Probabilities those mobility level is occurred
$C_{page}$	1	Unit page cost
$C_{reg}$	2	Unit registration cost
$P_{hit}^{area}$	0.75	Hit ratio of partial candidate area
$P_{hit}^{area}$	0.7	Hit ratio of candidate area
$P_{update}^{dir}$	0.25	Probability that direction-based registration is occurred

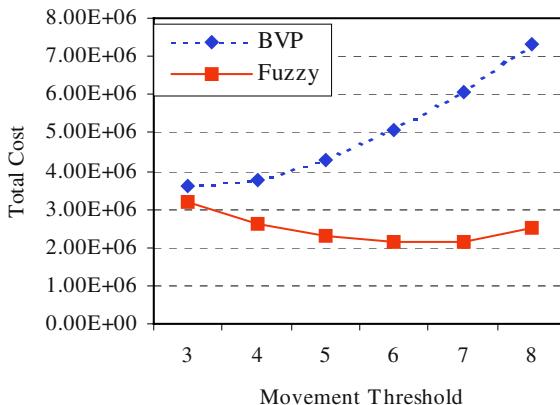
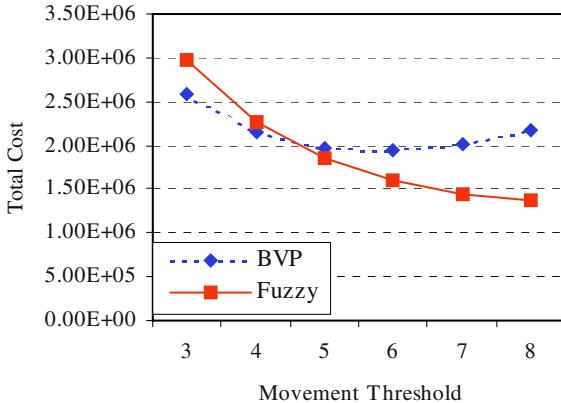
**Fig. 9.** The total cost vs. movement threshold ( $S=50000$ ).

Fig. 10 shows the total location management cost over movement threshold in the case that the number of calls attempted in the system is few. In the case that movement threshold is small, the performance of *Fuzzy* is worse than that of *BVP* because the amount of reduced paging cost by the fuzzy logic-based paging method is smaller than the amount of increased update cost by the direction-based registration. But in the case that movement threshold is larger, the performance of the *Fuzzy* is better than that of *BVP* because the fuzzy logic-based paging method reduces the paging cost.

## 5.2 Simulation

The performance of the fuzzy logic-based location management method is also to be evaluated by a simulation. In the simulation, we use the same values of the MT's velocity, cell radius, unit paging cost and unit registration cost as shown in Table 1. Table 2 shows the additional parameter values for the simulation.



**Fig. 10.** The total cost vs. movement threshold ( $S=10000$ ).

**Table 2.** Parameter values for the simulation.

Parameter	Value	Comment
$k$	4	Movement threshold
$m_d$	$\pi/3$	MT's mean velocity
$\sigma_d$	$\pi/18$	Standard deviation for MT's velocity
$T$	random (2, 9) min.	Call inter-arrival time

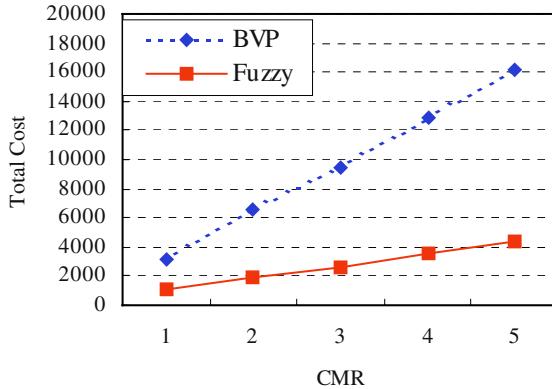
In Table 2, we assume that the MT's movement direction is a Gauss-Markov model. In continuous time, a stationary Gauss-Markov process [1] is described with a state differential equation:

$$\Theta(t+1) = -\beta\Theta(t) + \varpi(t) \quad (12)$$

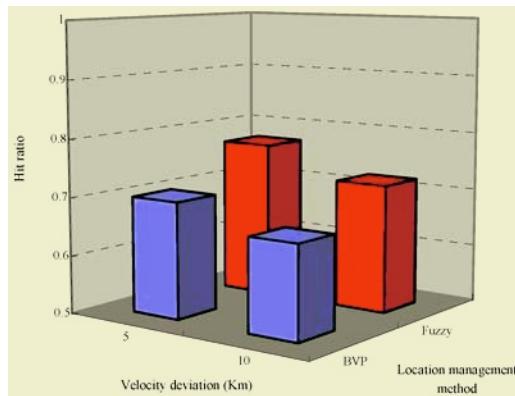
where  $\beta \geq 0$  determines the degree of memory in the mobility pattern and it is the correlation component.  $\varpi(t)$  is an independent, uncorrelated, and stationary Gauss process, with mean  $m_d$  and the standard deviation  $\sigma_d$ , represented as  $N(m_d, \sigma_d^2)$ . We set  $\beta$  to 0.1.

Fig. 11 shows the total cost by location management methods over call-to-mobility ratio (CMR) until the number of location updates is 100 times. The performance of the proposed *Fuzzy* method is better than that of *BVP* regardless of CMRs. Specially, we know that CMR is larger, the performance of our *Fuzzy* method is better because the paging cost is greatly reduced by paging only the cells within the partial candidate paging area.

Fig. 12 shows the hit ratio of *Fuzzy*'s partial candidate paging area and that of *BVP*'s candidate area over the velocity deviation, respectively. Both methods show that the smaller velocity deviation is, the higher hit ratio is. Although the size of *Fuzzy*'s partial candidate paging area is smaller than that of *BVP*'s candidate paging area, on the other hand, the hit ratio of *Fuzzy* method is higher than that of *BVP* method as much as about 7 percent. This result comes from the following reason. The *BVP* simply estimates the MT's velocity by the time taken between two movement-



**Fig. 11.** The total cost vs. CMR.



**Fig. 12.** Hit ratio of candidate area vs. velocity deviation.

based registrations, but the *Fuzzy* method estimates the more correct MT's velocity by the Error-based filter of EWMA.

Fig. 13 shows the number of location updates according to direction deviation and the number of movement-based updates. It is natural that the number of movement-based updates is occurred more frequently and direction deviation is large, the number of direction-based updates is increased. In the case that the velocity deviation is  $\pi/18$ , if the movement-based updates occurred 100 times, the direction-based updates occur 20 times, and if the movement-based updates occurred 300 times, the direction-based updates occur 77 times. Accordingly, when an MT occurs movement-based update, the probability that the MT occurred a direction-based update after the last movement-based update,  $P_{update}^{dir}$  is 0.2~0.256. Therefore, we set  $P_{update}^{dir}$  of the analytical model to 0.25.

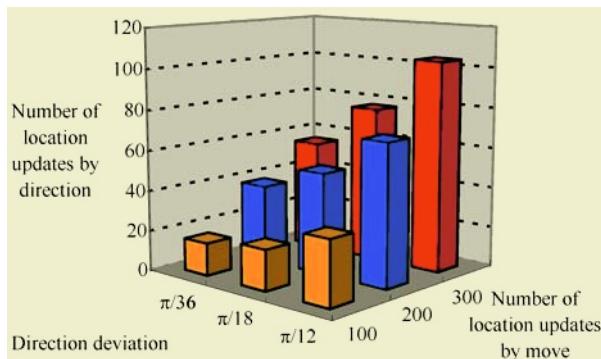


Fig. 13. Number of direction-based updates.

## 6 Conclusion

In this paper, we design and evaluate the fuzzy logic-based location management method on the basis of mobility information of an MT to reduce paging cost by predicting paging area, where MT's velocity and direction are selected as the core mobility information. In the proposed method, location update uses the area-based method that uses movement-based method together with direction-based methods, and location search uses the fuzzy logic-based selective paging method that is based on the basic fuzzy sets for the movement direction and the movement level of an MT. The *Fuzzy* method gets basic fuzzy sets for movement direction and movement level of the MT, selects a partial candidate paging area by using the fuzzy logic rule on the basis of the basic fuzzy sets, and pages only the cells within the selected candidate paging area. Accordingly, the paging cost of our method is greatly reduced.

The performance of the *Fuzzy* method is evaluated by an analytical model and by a simulation. As the results, the performance of the *Fuzzy* method is better than that of the BVP method. Specially, in case that the total number of calls attempted per hour is many and call-to-mobility ratio is large, the performance of the *Fuzzy* method is very excellent. We know that the size of *Fuzzy*'s partial candidate paging area is smaller than that of BVP's candidate paging area, but the hit ratio of the *Fuzzy* method is higher than that of BVP method.

## References

1. Ou, D-X., Lam K-Y., Dong D-C.: An Adaptive Direction-Based Location Update Scheme for Next Generated PCS Network. Proceedings of 2002 International Conference on Database and Expert Systems and Applications (2002) 413-422
2. Wan, G., Lin, E.: A Dynamic Paging Scheme for Wireless Communication System. Proceedings of the Third Annual ACM/IEEE International Conference on Mobile Computing and Networking (1997) 195-203
3. WAN, G., Lin, E.: Cost reduction in location management using semi-realtime movement information. Wireless Networks. 5(2) (1999) 245-256
4. Akyildiz, I. F., Ho, J. M.: On Location Management for Personal Communications Networks. IEEE Communications Magazine 34(10) (1996) 138-145

5. Wang, L-X.: A Course in Fuzzy Systems and Controls. Prentice-Hall (1997)
6. Ip, R. W. L., Lau, H. C. W. Chan, F. T. S.: An Intelligent Internet information delivery system to evaluate site preferences. *Expert Systems with Applications* 18(1) (2000) 33-42
7. Kim, M., Noble, B.: Mobile Network Estimation. In the Seventh ACM Conference on Mobile Computing and Networking (2001) 298-309
8. Tabbane, S.: An Alternative Strategy for Location Tracking. *IEEE Journal on Selected Areas in Communications* 13(5) (1995) 880-892
9. Hwang, H-W., Chang, M-F., Tseng, C-C.: A Direction-Based Location Update Scheme with a Line-Paging Strategy for PCS Networks. *IEEE Communications Letters* 4(3) (2000) 149-151

# A Next Generation Intelligent Mobile Commerce System

Eunseok Lee<sup>1</sup> and Jionghua Jin<sup>2</sup>

<sup>1</sup> School of Information and Communication Engineering, Sungkyunkwan University  
300 Chunchun Jangahn Suwon, 440-746, Korea  
[eslee@ece.skku.ac.kr](mailto:eslee@ece.skku.ac.kr)

<sup>2</sup> Department of Systems and Industrial Engineering, University of Arizona  
Tucson, AZ 85721, USA  
[jhjin@sie.arizona.edu](mailto:jhjin@sie.arizona.edu)

**Abstract.** Mobile Commerce(M-commerce) has some common critical problems such as the constraints on the small mobile device in both display and memory size, expensive charge system, limited contents, and so on. In this paper we propose some integrated solutions for solving these problems as follows: 1) *personalized contents providing* to increase the usability of the small device by reducing the amount of information transferred to the device with personalization policy. 2) a *middlet application* to support user's purchase activities such as product searching, ordering and settlement on the device with a minimum network connection. 3) *automated contents builder* to cope with the currently limited M-commerce contents.

The proposed system has been designed and implemented to demonstrate its effectiveness through experiments.

**Keywords:** mobile commerce, automatic contents builder, personalization, middlet application, intelligent agent, XML translator, recommendation, ontology server

## 1 Introduction

A rapid development in mobile communication technologies leads the diffusion of mobile devices such as cellular phone and PDA. It also leads the movement of general Internet services to mobile ones so that M-commerce has drawn big attention as the next generation commerce way. In world wide, however, although the spread of mobile device is enormous, the ratio of mobile internet services including M-commerce is not so high. The reasons might be derived from the followings: i) limited contents provided, that is, most of them are on the DB built for M-commerce only, ii) small display area and memory size in mobile device, and iii) high mobile call charge that increases with the amount used. These problems are commonly discovered from most commercial mobile services such as Wapeye[1], KTF's MagicN[2], SK Telecom's Nate[3] and Itouch[4].

In this paper we propose an intelligent M-commerce system that can overcome the above constraints. Concretely, for i), an automatic mechanism is built for transforming legacy online contents to mobile ones. The contents include the information on shops, products, and ordering forms of each shop for M-commerce. For ii), a user

based adaptively personalized filtering and display method is developed to reduce the amount of information transferred to mobile device. It is expected to overcome the constraints of mobile device with small display area and memory size. Finally for iii), a Middlet application is made to support user's all range of purchase activities like product searching, ordering and settlement on the mobile device with a minimum network connection. It is free from any kind of charge system and thus leading to a minimum call charge on using M-commerce. The proposed system has been designed and implemented to demonstrate its effectiveness through a number of experiments.

The paper is organized as follows: Section 2 describes the architecture and main functions of the proposed system. Section 3 shows the details of the module implementation and system evaluation. Section 4 provides conclusions and future work

## 2 Proposed System

### 2.1 System Architecture

The proposed system is intended to support all the purchasing activities, range from product search to payment, for M-commerce on the handheld devices by solving the problems mentioned above. Fig. 1 shows the architecture of the proposed system.

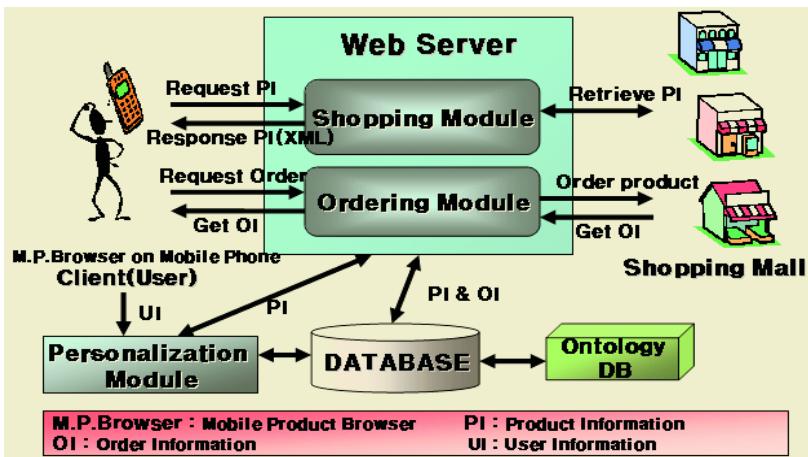


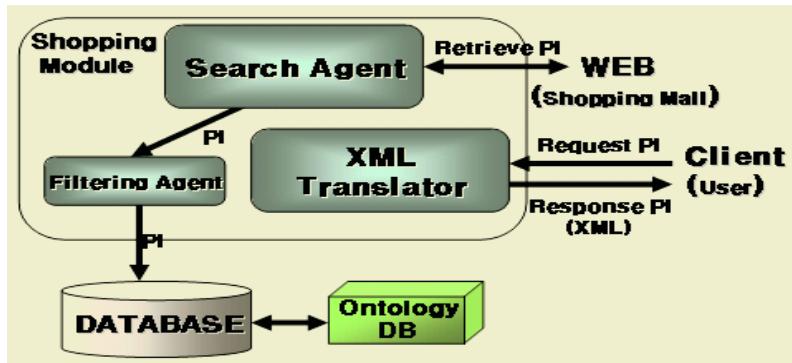
Fig. 1. Architecture of System.

The system consists of six main components as follows: (1) *Shopping Module* gathers and filters product information from shops on the Web. It translates product information into XML form that is then transmitted to the mobile device of client. (2) *Personalization Module* provides a user adaptive and individualized information with user data on preference and transactions. (3) *Ordering Module* gathers information related with order form of each shop and orders goods to the corresponding shop by filling in the order form. (4) *M.P.Browser (Mobile Product Browser)* is a middlet application which supports various types of products search on the mobile device. (5) *DATABASE* keeps information on goods that is gathered by the *Shopping Module*.

including user's personal information and transaction information. (6) *Ontology Server* keeps a standard category information on product categories that are used in a number of representative online shopping malls.

## 2.2 Shopping Module

This Module consists of Search Agent, Filtering Agent and XML Translator as shown in Fig 2.



**Fig. 2.** Architecture of Shopping Module.

*Search Agent* retrieves URL of shops that exist on the Web and extracts product information on each shop. The URL of shops is retrieved by analyzing shopping category of portal sites, and the agent extracts information on goods by visiting the page of each retrieved shop. The algorithm is applicable to the contents that are built with general web programming languages. The detailed algorithm of Search Agent is as follows (also shown in Fig. 3):

- (1) Finding the main page of shop. The first level category of products can be found on the main page in general.
- (2) Searching the first level categories. The category links of shops built with the web programming languages have some common characteristics as bellows: i) they are close by. ii) they have the same URL. iii) they have different values of variables. If the agent analyzes link tags on the page like this, it can obtain category information. The detailed method is as follows:
  - i) Finding link tags. (ex. `<a href = 'test1.asp?cate1=AA'>test</a>`)
  - ii) Separating tag and URL
  - iii) Extracting the links which have the pattern like '`filename.extension? {variable=value}+`'  
If the links have the same url of file such as "`filename.extension?`" and they are close by, it assumes that it is the first level (level 1) categories.
  - iv) Extracting category names. (Ex. `<a href=..>category names</a>`)

(3) Tracing the links of each category and searching offspring categories of each category repeatedly through the analysis of link patterns. The detailed method is as follows:

- i) Searching links of each category.
- ii) Repeating 3) until reaching to the products display page.
- iii) If the category does not have offspring categories anymore, the page of the link is assumed as the product display page.

(4) Visiting the product display pages, in which they have links with the product information and image pages. Also, it has the same URL of pages and variable values that are different. The name and price can be extracted from the products display page. It is more efficient than finding them from the detailed information pages of products. The products price and name can be searched by analyzing the HTML source between image links of products. The information of price can be searched by finding numbers that are located around the words like ‘price’, ‘won’, ‘\$’, ‘dollar’ and so on. Detailed information of goods can be obtained from the detailed information page of goods before filtering of items.

```

Step1. Visit store
Step2. Search main page
Step3. Search 1 level category
Step4. Now category = 1 level category
Step5. Save information of 1 level category(url, category name)
Step6. For each category do the following:
    Step6.1 Visit category page
    Step6.2 Search offspring category of current category
    Step6.3 If (offspring category)
        Step6.3.1 Save offspring category
            information
        Step6.3.2 current category = offspring
            category
        Step6.3.3 Do Step 6
    Step6.4 Else
        Step6.4.1 display page = current category
        Step6.4.2 Search goods information such as URL of the
            detailed page, goods name and price
        Step6.4.3 Visit the detailed page
            Step6.4.3.1 Search the detailed goods
                Information
            Step6.4.4 Save goods information

```

**Fig. 3.** Algorithm of Search Agent.

*Filtering Agent* filters goods information that is retrieved by Search Agent. Here the agent removes irrelative information on HTML tag such as tables, fonts, links, images, comments and so on.

*XML Translator* searches product information after receiving product search request from a client. And it converts the results of search to XML form and transmitted to the client. The category of goods is re-categorized by a standard category based on Ontology Server.

## 2.3 Ordering Module

Ordering Module consists of Order Agent and Retrieval Agent as shown in Fig 4. Retrieval Agent gathers information related with order from shops. The retrieved

information includes path of order, path of login and necessary filling in item when user orders. The order page has a link pattern like category. It visits the page of links and extracts information on order page by analyzing <form> tag of HTML. The page is an actual ordering page of shop. It obtains the information about variable transmitted in the actual order page. Usually, it can know the role of variables through the text in the front of <input> tag. The role of variables is decided by Ontology Server.

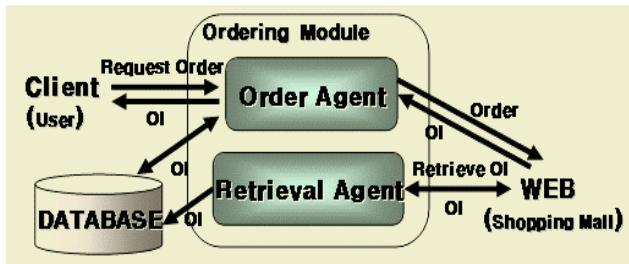


Fig. 4. Structure of Ordering Module.

*Order Agent* connects and orders the shop which sells relevant goods if a client requests an order. If the order is completed, the agent transfers the result to the client. Both personal information obtained from user's personal data and delivery information given from the user are used for ordering process.

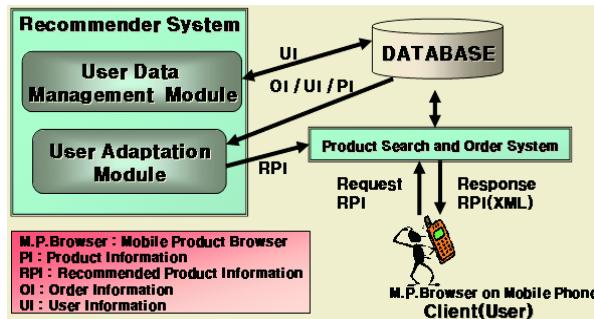
## 2.4 Personalization Module (Recommender System)

The Recommender System consists of User Data Management Module and User Adaptation Module as shown in Fig.5.

*User Data Management Module*. This module manages user's personal data such as user ID, name, age, sex, etc., which are necessary for settlement and login to each shop, and users historical purchase data about ordering and recommendation of goods.

*User Adaptation Module*. This module recommends goods with user's personal data and purchase data. It consists of two functional components, i.e., clustering module and mining module.

*Clustering Module* clusters customers based on the similarity of personal data of customers. ART2 algorithm is used for clustering users by Neural Net. The ART algorithm has the characteristic that can solve the Stability-Plasticity problem through unsupervised learning. The Stability-Plasticity problem is one of problems of existing Neural Net. The stability means the ability that keeps memory stably for patterns which learned before. The plasticity means ability that can process new pattern which has not studied. Therefore, it can organize by itself, and control the intensity of classification through the vigilance parameter. If the value of the vigilance parameter is set high, a specific and minute clustering is achievable. ART2 algorithm improves ART algorithm and has the capability to use the serial data[6]. This algorithm is shown in Fig. 6.

**Fig. 5.** Structure of Personalization Module.

Step1 . Begin with one cluster center  $w_1$   
 Step2 . Initiate  $w_1 = x$  (1) and  $|N_1| = 1$   
 Step3 . Loop through the data set X pattern-by-pattern  
     For each pattern do the following:  
     Step3.1 Present  $x(n)$  to the ANN .  
     Step3.2 Compute the winner .  
     Step3.3 Compare  $\|w_j^* - x(n)\|$  to the vigilance parameter  $p$  .  
         Step3.3.1 If  $\|w_j^* - x(n)\| \geq p$  then there is a resonance .  
             The weight  $w_j^*$  are updated according to  
              $w_j^* = [|N_j^*| w_j^* + x(n)] / [|N_j^*| + 1]$   
             And update the clock  $|N_j^*| = |N_j^*| + 1$   
         Step3.3.2 If  $\|w_j^* - x(n)\| < p$  then create a new node with  $w_j = x(n)$  and  $|N_j| = 1$

**Fig. 6.** Algorithm of ART2.

*Mining Module.* This Module selects the recommendation products by using Apriori-All algorithm[13]. It uses data about purchase information of customers who are in the same cluster. This data is clustered by *Clustering Module*. This algorithm relies on association rule to analyze the sequential purchasing patterns. The concept of this algorithm is as follow: i) there are a history of purchases :  $A \rightarrow B \rightarrow C$

- ii) someone purchases :  $A \rightarrow B$
- iii) probability that the user of ii) buys C is high

The points of association rule are to satisfy the criteria of the minimum support and minimum confidence. The main concept of Apriori Algorithm is as follows:

- i) Let  $I = \{i_1, i_2, \dots, i_m\}$  be a set of literals, called items.
- ii) Let  $D$  be a set of transactions, where each transaction  $I \subseteq T$  is a set of items such that  $T \subseteq I$ .
- iii) Let  $X$  be a set of items in  $I$ . A transaction  $T$  is said to contain  $X$ , if  $X \subseteq T$ .

An association rule is an implication of the form  $X \rightarrow Y$ , where  $X \subseteq I, Y \subseteq I$ .

Minimum confidence( $X$ ):  $Y$  holds with confidence  $c$  in transaction set  $D$ , if and only if  $c\%$  of transactions in  $D$  that contain  $X$  also contain  $Y$ .

Minimum support( $X$ ):  $Y$  has support  $s$  in transaction set  $D$ , if and only if  $s\%$  of transactions in  $D$  contain  $X \rightarrow Y$ .

The AprioriAll algorithm takes a serious view concept of transaction time. Both Transaction ( $A \rightarrow B$ ) and Transaction ( $B \rightarrow A$ ) include A and B, but it is different transaction. The main step of algorithm is as follow:

- i) Create a large item set that satisfy minimum support.
- ii) Create rules that satisfy minimum confidence using the large item set.

AprioriAll Algorithm is shown in Figure 7. The recommended goods information selected by *Recommender System* is translated into XML form by *PSOS*. The information of goods searched by user request is transmitted to the *M.P.Browser*.

```

Step1 . L1 = {large 1 - sequence}
Step2. for (k = 2; Lt-1 != 0 ; K++) do
    Step2.1 Ct = New candidates generated from Lt-1
    Step2.2 For each customer-sequence c in the database do
        Step2.2.2 Increment the count of all candidates
                in Ct the are contained in c
    Step2.3 Lt = Candidates in Ct with minimum support
Step3 . Answer = Maximal sequence in Ut Lt

```

**Fig. 7.** Algorithm of AprioriAll.

## 2.5 Middlet M.P.Broswer (Mobile Product Browser)

This Browser is a middlet application that is executed on mobile device. If user transmits query to our server(*PSOS*), the application connects the mobile device with the server through mobile network and makes the device to receive goods information written by XML form. Then the connection is closed and user can search goods on the mobile device without any connection with the server. The browser offers four types of search methods by parsing the transmitted XML file. First, the search of recommendable goods can search recommended goods. Second, directory search method can search goods by category in XML file. Third, price-based search method can search goods by price. Fourth, keyword-based method can search goods by keyword. When a user decides to buy a product, at this moment, the mobile device connects with the network server and it sends an order request to the server after user's input on personal information such as user ID and password, and delivery information. The connection is temporally cut again until it receives a result of the request from the server.

## 3 Implementation and Evaluation

The implementation environment is as follows: we employed IIS Web Server based on Windows2000 and MS SQL2000 for server side, and J2ME and Wireless Tool Kit as simulator [14] for client side.

### 3.1 System Implementation

Implementation of Shopping Module. Search Agent analyzes the shopping category page of large scale portal sites and extracts URL, name and category of each shop as

```

http://www.justbecause.com/=>Just Because...
http://www.800iloveyou.com/=>1-800-I-LOVE-YOU
http://www.flowersdirect.com/=>Flowers
http://www.originalflorist.com/=>Canada - Original Florist
http://www.season96.com/=>SeasonsSpecialties
http://www.botanica.com/=>Botanica
http://www.888-888-rose.com/=>1-888-888-ROSE
http://www.widdi.com/florist/american.html=>All American Florists
http://www.sugarlandflorist.com/=>Sugar Land Florist
http://www.compuflowers.com/=>CompuFlowers

```

**Fig. 8.** Search Results of Shop Information by Search Agent.

```

***** Goods Information *****
http://www.bestflowers.com/details.cfm?id=6=>29.99=>=Sweet and Simple
http://www.bestflowers.com/details.cfm?id=18=>69.99=>=A Walk in the Garden
http://www.bestflowers.com/details.cfm?id=28=>44.99=>=Golden Blaze
http://www.bestflowers.com/details.cfm?id=36=>79.99=>=Elegance In White
http://www.bestflowers.com/details.cfm?id=75=>59.99=>=Inspiration
http://www.bestflowers.com/details.cfm?id=93=>54.99=>=Memories of You
***** Goods Information *****
http://www.bestflowers.com/details.cfm?id=845=>34.99=>=Festival of Color
http://www.bestflowers.com/details.cfm?id=882=>94.99=>=Bright Future

```

**Fig. 9.** Search Results of Product Information on each Shop by Search Agent.

```

<font size=+1 face="arial" color="green"><b>It's a Girl</b></font> &#150;
<font size=2 face="arial" color="green"><b>FX3501-108</b></font>
<p>
<font size=2 face="arial" color="purple">Celebrate the arrival of a new baby girl with tulips and mini carnations perfectly placed in a cuddly teddy bear vase. (Container not available in all regions - will substitute similar quality.)</font>
<p>
<form action="addtocart.cfm" method="post">
<input type="hidden" name="ID" value="79">
Result => It's a Girl FX3501-108 Celebrate the arrival of a new baby girl with tulips and mini carnations perfectly placed in a cuddly teddy bear vase. (Container not available in all regions - will substitute similar quality)

```

**Fig. 10.** Filtering Result of Product Information by Filtering Agent.

shown in Fig.8. Then it collects information on products of each shop, such as URLs for each product, prices, and product names as shown in Fig.9.

*Filtering Agent* extracts essential products information by removing HTML tag from the information retrieved by Search Agent. Fig.10 shows a filtering result from complex HTLM source.

*XML Translator* searches the detailed information of goods and translates the information into XML form transmitted to client. The categories of goods are reorganized through the standard category defined in Ontology Server.

주소(D) | http://icoma.skku.ac.kr/ZZang/test2.asp?q=rose

```

<?xml version="1.0" ?>
- <Root>
  - <Recommend>
    - <Goods>
      <code>4567</code>
      <item>Congratulations</item>
      <name>Champagne Wishes</name>
      <price1>100000</price1>
      <storeno>11968</storeno>
      <store>Bestflower</store>
    </Goods>
    + <Goods>
    + <Goods>
  </Recommend>
  - <Cate>
    <Sub>Anniversary</Sub>
    <Sub>Basket</Sub>
    <Sub>Congratulations</Sub>
    <Sub>Love</Sub>
    <Sub>Mixed</Sub>
    <Sub>Others</Sub>
    <Sub>Sympathy</Sub>
    <Sub>Thank You</Sub>
  </Cate>
  - <List>
    - <Goods>
      <code>3897</code>
      <item>Sympathy</item>
      <name>Pink Single Rose</name>
      <price1>25000</price1>
      <storeno>12153</storeno>
      <store>8282flower</store>
    </Goods>
    + <Goods>
    + <Goods>
    + <Goods>
    + <Goods>
  
```

**Fig. 11.** Translated Products Information to XML.

**Implementation of Personalization Module.** *User Management System* manages user's personal data, login data for each shop, and historical purchase data. The user's personal data include user ID, name, gender, age, job, interest, phone-number, address, and settlement data for payment. The login data consist of membership ID and password for each shop. The purchase data include purchasing items, purchasing date, and purchasing user IDs per item.

*User Adaptive System* clusters users into related groups and recommends products according to the features of each cluster. We actually used real data on a specific shopping mall for experiments, which includes 240 users, 50 products items, and 550 purchasing data. Product search is commenced with user query, and the result is prioritized with purchasing preference, which includes preferences in shops and cost range.

**Implementation of Ordering Module.** *Retrieval Agent* gathers ordering information such as path for login, URL of order page, input items for ordering, and so on. *Order Agent* orders the corresponding shop automatically with user ID, password, and ordering data whenever a client requests, and returns the result to the client.

**Implementation of Mobile Product Browser.** *MP. Browser* is a middlet application that supports all range of purchasing activities. Fig.12 shows a use case for purchase with a real image. If a user inputs a keyword on the screen (Fig.12 ①) and presses search button, the user can connect to our server. After receiving XML file from the server, the network is disconnected. The browser offers various types of search methods by using data based on transmitted XML file (Fig.12 ②).



Fig. 12. M.P.Browser's work flows.

Directory search can search goods information by category. It displays goods information of relevant category (Fig.12 ③). Fig.12 ④ is a result when ‘Anniversary’ is selected. And it offers searching information of goods by price and keyword (Fig.12 ⑤, ⑥). M.P.Browser connects with the server when the user requests an order. The user inputs information such as user’s system ID, password and delivery. The browser transmits the information to the server. The browser displays the result of order from the server after the ordering process and disconnect the connection again(Fig.12 ⑦).

### 3.2 System Evaluation

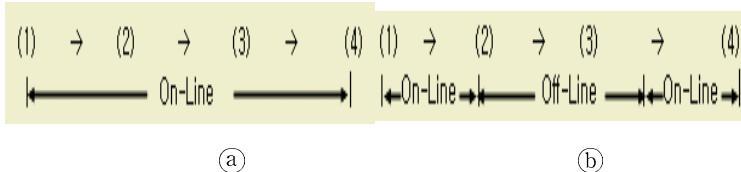
Our experiments have intended to show the followings: one is to show the possibility of automatic and dynamic extensibility of general e-commerce contents for m-commerce. Another is to show the possibility of minimizing the connection time for m-commerce.

**Automatic Extensibility of Contents for MC.** We have actually applied our system to 100 shops selected randomly on the Internet. Through the experiment, at 62 shops among 100, we have confirmed that the proposed system could gather, extract the shop’s information including the category structure, products information, and order form etc., and store the information to our database for m-commerce without any load. The other 38 shops were excluded due to their too individual and poor structure. By this system, users for m-commerce can purchase on the unrestricted contents with

an impression of e-commerce. The mobile service provider can reduce the burden of building the contents. The system offers product information in XML form so that it can be applied to various platforms.

**Minimizing of Connection Time for m-Commerce.** The mobile call charge increases with the amount used. That is, the charge is imposed with connection time to mobile network. That is a big obstacle to activation of m-commerce, particularly in South Korea.

A general work flow for m-commerce is as follows: (1)Request products information → (2)Browse the provided information → (3)Input information for Order → (4)Order



**Fig. 13.** Comparison of a general MC with our system in connection time.

The connection time of the general M-commerce is as shown in Fig.13 (a). We could largely reduce the time as shown in (b), because users especially spend a lot of time at work (2). Therefore, the application supports efficient m-commerce by offering an economic and convenient interaction way.

## 4 Conclusions

In this paper, we proposed an integrated solution for intelligent M-commerce. One merit of the proposed system is its functionality of translating the contents developed for conventional e-commerce to m-commerce automatically and dynamically. By this, we could resolve the problem of restricted contents of M-commerce. Another merit is its effective mechanism for reducing the connection time for M-commerce by providing a middlet application, which works for searching, ordering, and settlement at mobile devices. Furthermore, we provided some personalized services, that is for reducing the information transferred to the mobile device and improving the usability on the small display area. We have actually designed and implemented the system, and evaluated its effectiveness partially in practice. The further efforts will be devoted in future to enhancing the reliability of personalization policy.

## Acknowledgement

This research was partially supported by BK21 project in Korea and a support program in SKKU.

## Reference

1. <http://www.wapeye.net>
2. <http://www.nate.com:81/>
3. <http://www.magicn.com/>
4. <http://www.itouch017.com>
5. J. Yang, H. Seo, and J. Choi, "MORPHEUS: A Customized Comparison Shopping Agent", 5th International Conference on Autonomous Agents (Agents-2001), pp. 63-64, Montreal, Canada, 2001.
6. Thorsteinn S. "Lecture Note: Self-Organizing Algorithms", <http://www.hh.se/staff/denni/>
7. IBM Almaden Research Center, "Mining Sequential Patterns", <http://www.cs.duke.edu/~geng/>
8. Glushko, R.J., Tenenbaum, J.M., Meltzer, B.: An XML framework for agent based E-commerce. Communications of the ACM March 1999 Volume 42 Issue3
9. Tveit,A.: Peer-to-peer based recommendations for mobile commerce. Proceedings of the first international workshop on Mobile commerce July 2001
10. Papazoglou, M.P.: Agent-oriented technology in support of e-business. Communications of the ACM April 2001 Volume 44 Issue 4
11. Tsalgatidou, A., Veijalainen, J.: Mobile Electronic Commerce: Emerging Issues In Proceedings of EC-WEB, pages 477-486, September 2000
12. Varshney, U., Vetter, R. J., Kalakota, R.: Mobile Commerce: A New Frontier. IEEE Computer, 33(10):32-38, 2000
13. Purdom, P., Gucht, D.V.: Average Case Performance of the Apriori Algorithm Technical reports from Indiana University Computer Science Department
14. <http://java.sun.com>

# **Indexing XML Data for Path Expression Queries**

Gongzhu Hu and Chunxia Tang

Department of Computer Science, Central Michigan University  
Mount Pleasant, MI 48859, USA  
hu@cps.cmich.edu

**Abstract.** Many XML query languages use regular path expressions to query XML data. Retrieval of XML data is like retrieval of relational databases, but the difference is that relational data are stored in 2-D flat tables whereas XML data are organized in a tree-like structure. Hence, fast tree traversal is a key to XML query processing. This is commonly accomplished using indexing. In this paper, we present a design and implementation of an efficient and quick solution to indexing XML data. Our approach is based on a numbering scheme that encodes the XML elements for not only indexing to the elements but also for quick determination of the ancestor-descendant relationship between elements in the tree hierarchy. This approach also allows efficiently inserting and updating the index system, which is an improvement over several existing XML tree node numbering methods.

## **1 Introduction**

Since XML (eXtensible Markup Language) was adopted by World Wide Web Consortium as the standard format and language for data exchange on the Web, researchers and developers have focused their attention to find efficient ways to query and retrieve XML data. Just like retrieval of relational databases, fast retrieval of XML data is the key to the success of the new XML technology. A common approach to achieve this is to use indexing techniques. Indexing is well developed technique for relational databases, but yet to mature for semi-structured data sets like XML. According to Robert Luk, et. al. [13], there are three broad approaches for XML document retrieval and indexing: database-oriented approach, information retrieval-oriented approach, and hybrid approach. The database-oriented approach is to actually store data in relational databases but present in XML rendered XSL stylesheet format. So conversion must be done between XML documents and the storage in the database. Researches have been published in this area, such as [16,18,21]. There are also some DB-XML conversion tools [19] available that do the translation between DB and XML. The information retrieval-oriented approach directly applies indexing to the retrieval of XML documents that are considered to be text documents. Index can be built on tags as if the tags were index terms. Particular words in certain tags may be treated differently based on the nesting-relationships between the tags. The hybrid approach combines some popular techniques and often creates XPath indexes for search. Sometimes multi-level indexes are built.

As we know that an XML document is structured as a tree. Each node in the tree represents either an element or an attribute. Leaf nodes contain the actual data. To retrieve data from an XML document is essentially a tree search, which is generally

slow. Indexing to the tree nodes is to speed up the search. However, searching XML documents often requires meeting certain condition that involves a sequence of nodes along some paths in the tree. That is, the retrieval algorithm needs to keep track of the ancestor-descendent relationships while walking through the nodes. To facilitate meeting this requirement, indexing methods have been developed based on the concept of coding (or numbering) of XML nodes. The basic idea of node numbering is that each node is assigned a unique integer number in such a way so that it takes constant time to determine the ancestor-descendent relationship of any two nodes.

Some numbering systems have been proposed in the literature, but none has addressed the problem when new nodes are inserted into the tree, or at least they didn't provide a good solution to the problem. We have developed a new numbering scheme that is flexible enough to allow insertions without re-numbering the nodes that have already been numbered. Based on this numbering scheme, we also developed an XML indexing method to improve the performance of retrieving XML data for XML path expression queries. Our method stores XML data in a B+ tree and builds indexes based on the numbering scheme. Given a name from the query, which can be the name of an element or an attribute, the index structures can efficiently find the code number of the name. Conversely, given a code number, the indexing method can efficiently find all the names in the XML hierarchy that are the same as the one with the code number. Our method is tested using queries expressed in path expressions according to XPath [3].

## 2 Related Work

In recent years, researchers and developers have developed many indexing schemes for XML data. Each of these schemes and methods is aimed to improve the performance of query processing by creating indexes at various processing stages. The indexing strategy proposed by J. McHugh, et. al. [14] at Stanford University as part of the Lore project presented a general framework for index values in terms of automatic type coercion. It provides four types of indexes: value index, type index, link index and path index. These indexing approaches were applied in top-down, bottom-up, as well as hybrid query plans and the results showed a significant speedup.

Flavio Rizzolo and Alberto Mendelzon of the University of Toronto proposed an indexing method that synthesizes ideas from object-oriented path indexes and extends to semi-structured realm of XML data [15]. As part of the ToXin (Toronto XML Engine) project, their method uses an edge-labeled graph to model semi-structured data, in which nodes correspond to objects or values and edges represent elements or attributes. With this model, they use path index to summarize all paths in the database and value index to support predicates over values.

A 3-D bitmap indexing method for XML documents was proposed by Yoon, Raghavan and Chakilam [20]. In their method, a collection of XML documents is represented as a 3-D data structure: XML document, element path, and terms and words. They defined a BitCube index scheme to partition documents into clusters for efficient retrieval. The 3-D data structure allows operations, such as slicing on each dimension and dicing on cells, to provide easy and quick access to the information.

Li and Moon [12] developed a novel approach for indexing and querying XML data for path expressions. The idea is this: A path expression query is decomposed into basic

expressions, each of which corresponds to a simple unit in the expression and can be easily processed. The results of the simple expression are joined at the later stages. The join conditions are determined by the parent-child or ancestor-descendant relationships of the nodes in the XML tree. They proposed a numbering scheme for the nodes in the XML tree that can be used to easily determine such ancestor-descendent relationships. This numbering scheme leaves limited room for addition/deletion/updating of nodes in XML documents. They build element index, attribute index, structure index, and name index. The values are stored in a value table. Once the data satisfying certain conditions are retrieved using the indexes, joins (element-element, element-attribute, Kleene closure) are performed if certain ancestor-descendant relationships are met to produce the results of the query. Their experiments indicated that the performance on large XML documents was significantly improved over conventional tree search approach. We will discuss their numbering method later.

One of the difficulties of building indexes for any data is the update of data items. This is true for traditional relational databases and certainly true for semi-structured data as well. Most often, once some data items are updated or new items are added, the index entries also need to be updated. To overcome this difficulty, one approach is to identify the index entries that are affected by the data change and update only those entries. That is, to reduce the number of updates to the minimum. Kha, Yoshikawa, and Uemura proposed an indexing method [10] that records the “relative region coordinates” (RRC) of elements in an XML document. When the element n is updated (adding several bytes to the value, for example), only n’s right sibling and its ancestors and their right siblings need to update their RRCs. Those tree nodes (in some sub-trees) whose indexes (RRCs) are to be updated together are grouped and stored on the same disk block(s) to reduce the disk I/O. They developed algorithms to identify and construct block sub-trees.

Brain Cooper et al. proposed an indexing method, called Index Fabric [4], that encodes paths through semistructured data as simple strings and inserts those strings into a special index that is highly optimized for long and complex keys. Their method manages two types of paths, raw paths to accelerate ad hoc queries and refined paths to better optimize particular queries. The Index Fabric is built on top of a commercial relational DBMS. Their experiments showed that the indexing approach yields significant performance improvement over using the RDBMS’s native indexes for semistructured data.

Daniel Egnor and Robert Lord of XYZFind Cooperation suggested several ways to improve precision and recall for semantically structured XML document retrieval [6]. They use an index structure that is a modification of the classic “inverted index.” The modified inverted index specifies the association between keywords, XML path specifications where those keywords appear, and locations. Additionally, rather than merely storing the offset from the beginning of the document, this index records the XML address (a sequence of numbers representing which outer elements contain this content) of each word.

Several XML query engines have been developed and published. For the XML Text Search Engine, there are XQEngine [9], Amberfish [8], SIM [17], Emily Solutions Framework (MLE) [7]. XQEngine is a full-text search engine for XML documents. Utilizing XQuery [2] as its front-end query language, it lets the user interrogate collections

of XML documents for Boolean combinations of keywords, much as Google and other search engines let you do for HTML. Amberfish supports both XML full text search with Boolean and phrase matching, as well as arbitrarily structured queries with hierarchical results. SIM is the Structured Information Manager. It uses XML and full-text indexing to provide a powerful combination of database searching and text retrieval. MLE includes a search engine for XML, flat files and relational databases. It is a very high level language that is specifically targeted to the processing of markup documents, such as HTML and XML. For the XML Structured Query Engine, there are XDisect [22], Xset [23] and UC Berkeley's Cheshire [11], among others. XDisect indexes and retrieves arbitrary XML data structures and supports flexible queries. Xset is an XML database and high performance search engine library with a simple tag-oriented query language, expressed in XML itself. UC Berkeley's Cheshire system indexes and searches structured data including XML, SGML and MARC records, as well as full-text data files.

These are only some of the recent work. As we mentioned before, this is still a field of great research interest and we would like to develop our own index engine as a basic platform for further research. The major contribution of our work is the new numbering scheme for coding XML nodes to facilitate fast retrieval of XML data sought by XPath queries.

### 3 Numbering Scheme

XML query is the combination of value search and structure search. To process both value and structure searches on the XML database, it is critical to quickly determine the ancestor-descendant relationship between XML elements as well as fast access to the desired data.

XML data components are usually structured in a tree, where elements, attributes and text data are represented by nodes, and parent-child node pairs represent nesting between XML data components. To speed up the process of XML queries, it is important to improve the speed of structure searching. In turn, it is critical to quickly determine ancestor-descendant relationship between any pair of objects in the hierarchy of XML data. A common approach is to assign the nodes in the tree unique codes (numbers) and the ancestor-descendant relationship is quickly determined from the codes.

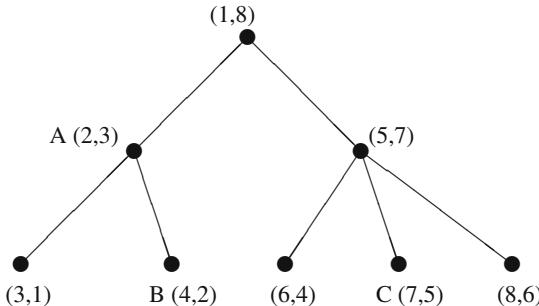
#### 3.1 Dietz Approach

One of the early observations was Dietz's numbering scheme [5]. He used tree traversal order, pre-order and post-order, to determine the ancestor-descendant relationship between any pair of tree nodes. His proposition was the following:

*For two given nodes  $x$  and  $y$  of a tree  $T$ ,  $x$  is an ancestor of  $y$  if and only if  $x$  occurs before  $y$  in the preorder traversal of  $T$  and after  $y$  in the post-order traversal.*

For example, consider a tree in Fig. 1, in which the nodes are annotated by Dietz's numbering scheme. Every node is numbered with a pair of preorder and post-order numbers. For example, node A (2, 3) is an ancestor of node B (4, 2), because node A appears before node B in the preorder number ( $2 < 4$ ) and after node B in the post-order number ( $3 < 2$ ). However, nodes A and C do not have this relationship because their

numbers do not satisfy the condition. The advantage of his approach is that the ancestor-descendant relationship can be determined in constant time. However, the disadvantage is the limited flexibility. That is, for each updating and inserting, the preorder and post-order numbers have to be recomputed for many tree nodes.



**Fig. 1.** Dietz's numbering scheme.

### 3.2 Li and Moon Approach

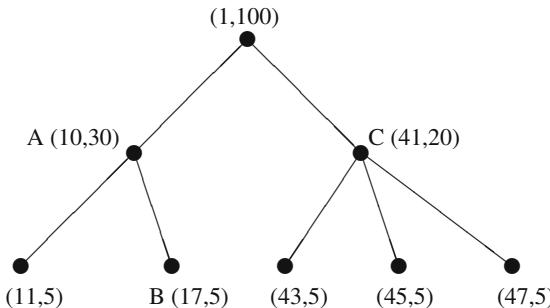
To solve the updating problem, Li and Moon proposed a numbering scheme [12] based on Dietz's idea. They used an extended preorder and a range of descendants. The proposal numbering scheme associates each node with a pair of numbers  $\langle \text{order}, \text{size} \rangle$ , where order is an extended preorder and size is a range of descendants, as follows:

- For a tree node  $y$  and its parent  $x$ ,  $\text{order}(x) < \text{order}(y)$  and  $\text{order}(y) + \text{size}(y) = \text{order}(x) + \text{size}(x)$ .
- For two sibling nodes  $x$  and  $y$ , if  $x$  is a predecessor of  $y$  in preorder traversal,  $\text{order}(x)+\text{size}(x) < \text{order}(y)$ .

Then, for a tree node  $x$ ,  $\text{size}(x) = \sum(\text{size}(y))$  for all  $y$ 's that are direct children of  $x$ . Their idea is to make the numbers far apart to make room for insertions. Their proposition was the following:

*For two given nodes  $x$  and  $y$ ,  $x$  is an ancestor of  $y$  if and only if  $\text{order}(x) < \text{order}(y) = \text{order}(x) + \text{size}(x)$ .*

Fig. 2 shows the numbered nodes of a tree based on Li and Moon's numbering scheme. Each node is associated with a pair of  $\langle \text{order}, \text{size} \rangle$ . In the tree, we can tell the node A (10, 20) is an ancestor of node B (17, 5), because  $\text{order}(B)$ , 17, is between  $\text{order}(A)$ , 10, and  $\text{order}(A)+\text{size}(A)$ , 10+30, that is  $10 < 17 < (10+30)$ . An obvious advantage of the approach is that it could not only determine the ancestor-descendant relationship in constant time, but also provide some level of flexibility for inserting and updating. However, this numbering method assigns the order and size to a node statically. Once assigned, they are fixed and can only allow a fixed number of insertions without re-numbering the nodes. Although there is no need to change the value of *order* that is the preorder value of the node (considering insertions are in fact appendings to the right of the siblings at each level), we would need to change the value of *size* in the pair.



**Fig. 2.** Li and Moon’s numbering scheme using  $\langle \text{orde}, \text{size} \rangle$  pair.

Of course, we can assign far-apart numbers to provide sufficient room for insertions. However, deciding the size of a node is not arbitrary nor is it trivial. The problem of Dietz’s approach still exists if the distance of two nodes has been used up.

### 3.3 Proposed New Numbering Scheme

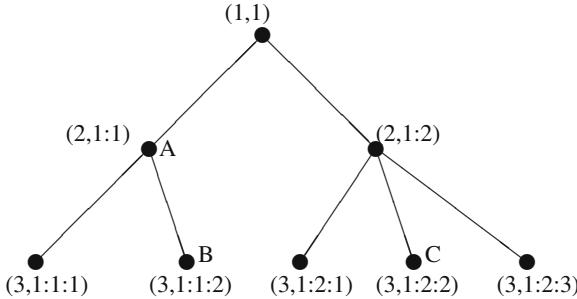
To solve the Dietz’s problem, we propose a new numbering scheme that associates each node with a pair  $(k, c)$ , where  $k$  is the level of the node in the tree, and  $c$  is of the form  $c_1 : c_2 : \dots : c_k$ . Let  $y$  be a node at tree level  $k$  and the  $i$ th child of its parent  $x$ . The code of  $y$  is  $(k, c(x) : i)$ , where  $c(x)$  is the  $c$  part of  $x$ ’s code. Here is the algorithm of assigning the  $(k, c)$  pair to the nodes:

1. The root node has the code  $(1, 1)$ . That is, the root is at level 1 and  $c(\text{root}) = 1$ .
2. At level  $k$  ( $k > 1$ ), the  $i$ th sibling node  $A$  is assigned the code  $(k, c(p) : i)$  where  $p$  is the parent of  $A$ .
3. At level  $k$ , assign the same serial number to all the sibling attributes so that they have the same code.

Let’s look at the example shown in the Fig. 3. The nodes are assigned their codes according to the above algorithm. Node  $A$ ’s code is  $(2, 1 : 1)$  and all  $A$ ’s children have a code of the form  $(3, 1 : 1 : k)$ . That is the first 2 segment  $c$  values of node  $B$ , which is 1:1, is the same as  $c(A)$ , and we know that  $A$  is an ancestor of  $B$ . On the other hand, node  $C$  is the same level with node  $B$ . However, the first 2 segment  $c$  values of node  $C$  is 1:2 which is different from the  $c(A)$ , and  $A$  is not the ancestor of  $C$ .

This numbering scheme allows insertions of new nodes without re-numbering existing nodes, considering the fact that for XML elements, insertions are indeed appendings to the right of the siblings. It is not hard to see that each node inherits the  $c$  value from its parent and concatenates its own ordinal number. Thus, this approach guarantees that for a pair of tree nodes  $x$  and  $y$ , the first  $n$  segments  $c$  values of  $y$  is the same as the  $c(x)$  if  $x$  is the ancestor of  $y$ . This observation leads to the following lemma:

**Lemma 1.** *For two given nodes  $x$  and  $y$  of a tree  $T$ ,  $x$  is on level  $n$  with code  $(n, c_1 : \dots : c_n)$  and  $y$  is on level  $m$  with code  $(m, d_1 : \dots : d_m)$ .  $x$  is an ancestor of  $y$  if and only if  $m > n$  and  $c_1 : \dots : c_n = d_1 : \dots : d_n$ . That is, all number segments of  $c(x)$  equal to the first  $n$  number segments of  $c(y)$ . In order words,  $c(y)_n \text{ XOR } c(x)_n = 0$ .*



**Fig. 3.** Numbered nodes with our new numbering method.

*Proof.* Let  $n = 1$  and  $m = n + 1 = 2$ . The algorithm assigns  $(1, 1)$  to the root node  $x$ . For any child node  $y$  of  $x$ ,  $y$ 's code will be  $(2, 1 : k)$  where  $k$  is the unique ordering number of  $y$  among its siblings. The Lemma obviously holds. Assume that the Lemma is true for  $m - n = k$  for arbitrary nodes  $x$  at level  $n$  and  $y$  at level  $m$ . Let's consider  $m - n = k + 1$  where  $z$  represents any node at level  $m - 1$  and also a descendent of  $x$ . If  $y$  is a child of  $z$  and hence a descendent of  $x$ ,  $y$  would be assigned  $(m, d_1 : \dots : d_{m-1}d_m)$ , where  $d_1 : \dots : d_{m-1}$  is inherited from  $z$  and the first  $n$   $d_i$ 's are the same as the  $x$ 's code according to the induction hypothesis. If  $y$  is a child of  $w$ , which is at level  $m - 1$  and not a descendent of  $x$ , the first  $n$  numbers,  $s_1 : \dots : s_n$  of  $w$  (also of  $y$ ) would be different from that of  $x$  according to the hypothesis.

### 3.4 Performance Analysis

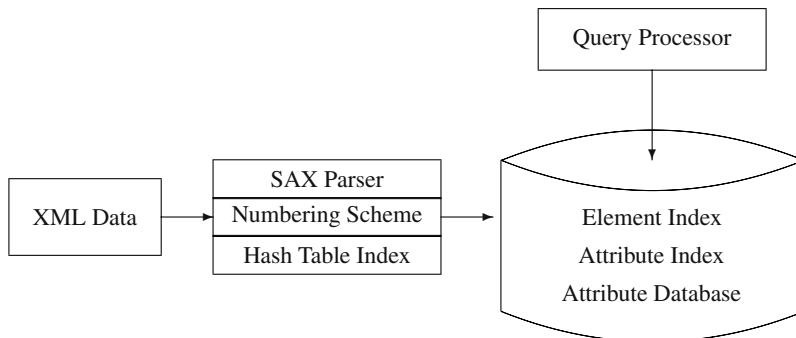
Theoretically, it takes constant time to determine if a node  $x$  is an ancestor of another node  $y$ . It only involves a masking operation (to get the first  $n$  number segments of  $y$ ) and an XOR operation, both are  $O(1)$  operations. In addition, compared with other two approaches, we use the segments for each part of numbering which increase the flexible and dynamic updates of XML data. When a new node is added at level  $n$ , we simply assign  $(n, c(p) : i)$  to the node, where  $c(p)$  is the  $c$  value of its parent  $p$  and  $i$  is the ordinal number of the new node among its siblings. There is no need to re-number the codes of existing nodes.

In implementation,  $n$  is an integer;  $c = c_1 : \dots : c_n$  can be defined as an integer that is concatenation of the  $c_k$ 's, or a vector of integers. Using an integer to represent  $c$ , the comparison operation is guaranteed to be  $O(1)$ , but it imposes a limit to the size of the tree. An integer on a common computer is 32 bits that can hold  $L$  levels with  $2^{32}/L$  nodes at each level. For example, if a tree has 5 levels, each node may have 64 (26) children. Although these numbers do not look large enough, the idea can be extended to using a vector of integers. The concatenation of the integers in the vector can be thought of as a very long integer to provide more space. For example if we use 4 integers (total of 128 bits), we can represent a code  $c_1 : \dots : c_{16}$  of 16 levels for a node that has up to 256 ( $2^{128}/16 = 28$ ) children, which is normally large enough to handle most XML documents. Here we fix the number of integers in the vector to guarantee the

$O(1)$  operation time. To the extreme, if we really want to allow “unlimited” levels and siblings, we can use a single integer for each  $c_k$ . Then, we would have a dynamically size-adjusted vector of integers to provide total flexibility to allow arbitrary insertions. In this case, the size of the vector would be  $n$ , the level of the node in the tree, which is in average  $O(\log_d N)$  where  $N$  is the number of nodes in the tree and  $d$  is the average degree of fan-out. This is obviously worse than  $O(1)$ . However, because the depth of an XML tree is defined in DTD or XSchema before the data are created, the number of levels of the tree is fixed and does not increase as the size of the tree increases (nodes are inserted at fixed levels while the number of levels does not change), the performance can still be considered constant.

## 4 Index and Data Organization

We designed a query system that uses the numbering method to build indexes to the elements, attributes, and texts (data at the leaf nodes). Search can be performed based on the name of an element, an attribute, or the actual data. The index structure of this approach is composed by several components, shown in Fig. 4.

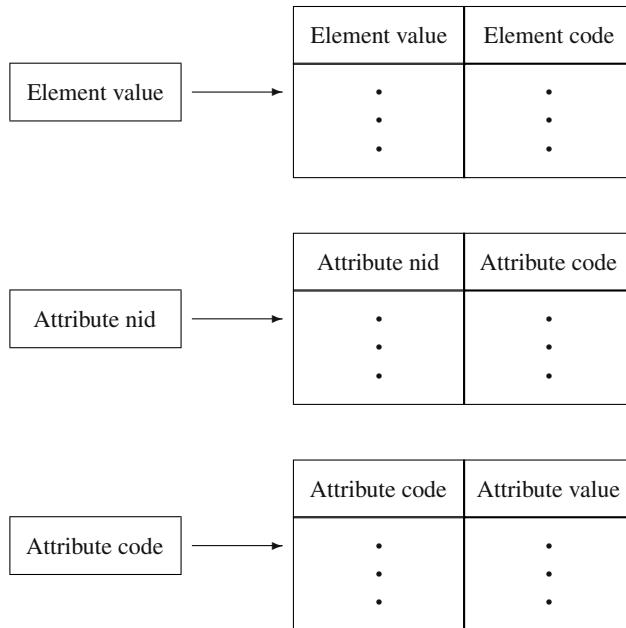


**Fig. 4.** Overview of indexing and querying structure.

The name string of element is stored in the element index table which has the element value as the index key and element numbering as the data. The attribute structure is a little different because the attribute name string is stored in the attribute database file with the attribute numbering as the index key. The hash table will give the attribute nid as entry to attribute index file which would have attribute numbering as data. All of these index files and database files are implemented as a B+-tree. An element or attribute can be uniquely identified by its numbering in the system. The text part would have the same structure with attributes. The only difference is that the text has the same numbering code as the element which it is associated with. The element index and the attribute index components are shown in Fig. 5.

Each element index file uses the element value as the entry key for each record, which includes the level number and code number, as shown in Fig. 6.

Fig. 7 is a similar structure for the attribute index file that uses the attribute nid as the entry key.



**Fig. 5.** Element and attribute indexing structures.

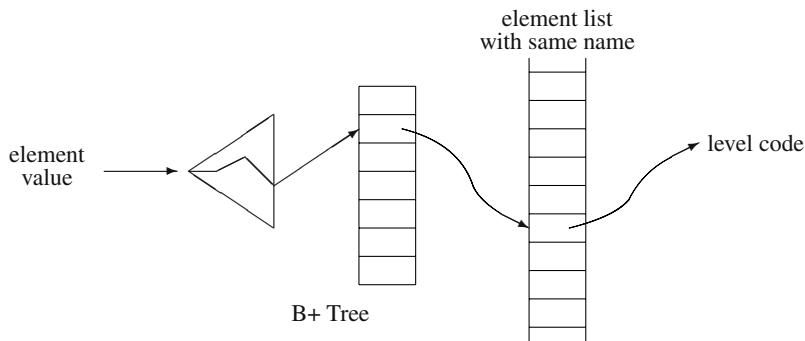
Each attribute database file uses the attribute code number as the index key and has the attribute value in the record as shown in Fig. 8.

The vector would hold all the items with the same value in each list. Each item in the vector could have the variable-length record associated with the element or the attribute. In the vector, the record is sorted by their code number.

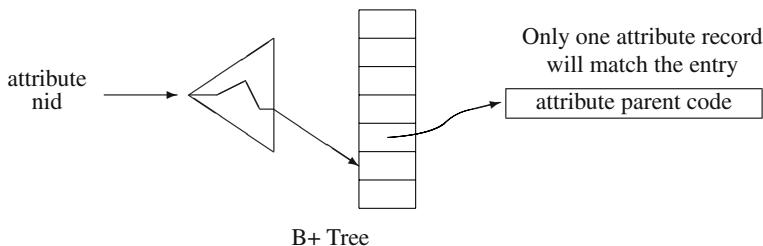
## 5 Example of Encoding and Index File Structures

Let's consider an sample XML document (bookstore.xml) and the numbering of the its elements.

```
<?xml version ='1.0' encoding='utf-8'?>
<!-- A Sample set of Book Store -->
<bookstore>
    <!-- Title Slide -->
    <book title="Fundamentals of C++" author="Nance Lambert">
        <unit1>
            <chapter1 topic1="Computer History" topic2=
                "Computer Architecture"> </chapter1>
            <chapter2 topic="Basic Concept"> </chapter2>
        </unit1>
        <unit2>
```



**Fig. 6.** Element index structure.



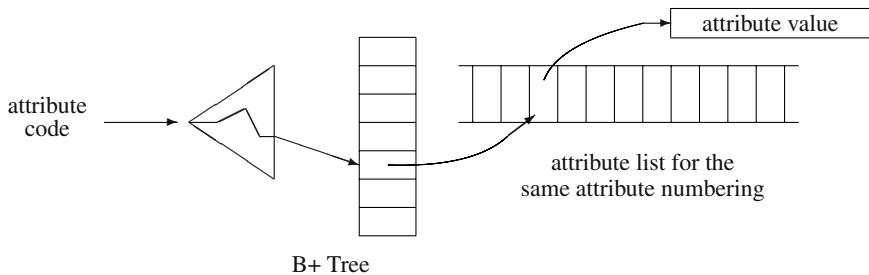
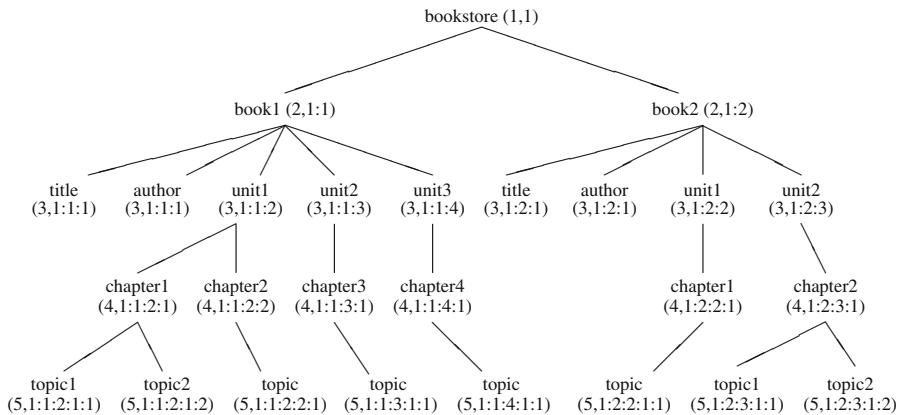
**Fig. 7.** Attribute index structure.

```

<chapter3 topic="User Defined Function"> </chapter3>
</unit2>
<unit3>
    <chapter4 topic="Selection Statements"> </chapter4>
    </unit3>
</book>
<book title="Cryptography and Network Security"
      author="William Stallings">
    <unit1>
        <chapter1 topic="Basic Concept"> </chapter1>
    </unit1>
    <unit2>
        <chapter2 topic1="Encrypt Model" topic2=
                  "Decrypt Model"> </chapter2>
    </unit2>
</book>
</bookstore>

```

In this example, we use italic font for elements and bold font for attributes. According to the proposed numbering scheme, the tree nodes and their codes are shown in Fig. 9 below. We will use this example later.

**Fig. 8.** Attribute database structure.**Fig. 9.** Numbered nodes of the sample XML data.

Based on the index file structures discussed before, we build separate indexing files on elements, attributes, element numbering and attribute numbering. We also have three indexing files corresponding to element numbering, attribute value and attribute numbering. Element index file has the element nid (node id) as the entry where nid the numbering code used as the index built on the element value. We use a hash function  $h$  for the index number (i.e. element nid):

```
element nid = h(element name)
```

For the sample XML document, the element index file contents are shown in Table 1 below.

The attribute index file has the attribute nid as the entry where nid is the index number built on the attribute value. The attribute nid is similar to the element nid. The attribute index file for the sample XML data file is shown in Table 2.

Table 3 shows the contents of the attribute database file. It uses the attribute parent numbering code as the entry where attribute numbering code is obtained from the attribute indexing file. For a given attribute numbering code, the attribute database file provides all the attribute values of the same code.

**Table 1.** Element index file.

<b>Element Value</b>	<b>nid (element numbering code)</b>
Hash(bookstore)	(1,1)
Hash(book)	(2,1:1)
Hash(book)	(2,1:2)
Hash(unit1)	(3, 1:1:2)
Hash(unit1)	(3, 1:2:2)
Hash(unit2)	(3, 1:1:3)
Hash(unit2)	(3, 1:2:3)
Hash(unit3)	(3, 1:1:4)
Hash(chapter1)	(4, 1:1:2:1)
Hash(chapter1)	(4, 1:2:2:1)
Hash(chapter2)	(4, 1:1:2:2)
Hash(chapter2)	(4, 1:2:3:1)
Hash(chapter3)	(4, 1:1:3:1)
Hash(chapter4)	(4, 1:1:4:1)

**Table 2.** Attribute index file.

<b>Attribute nid</b>	<b>Attribute numbering code</b>
Hash(title “Cryptography and Network Security” )	(3, 1:2:’1)
Hash( title “Fundamentals of C++”)	(3, 1:1:’1)
Hash(author “Nance Lambert”)	(3, 1:1:’1)
Hash(author “William Stallings”)	(3, 1:2:’1)
Hash(topic “Basic Concept”)	(5, 1:1:2:2:’1)
Hash(topic “Basic Concept”)	(5, 1:2:2:1:’1)
Hash(topic “Selection Statements”)	(5, 1:1:4:1:’1)
Hash(topic “User Defined Function”)	(5:1:1:3:1:’1)
Hash(topic1 “Computer History”)	(5, 1:1:2:1:’1)
Hash(topic1 “Encrypt Model”)	(5, 1:2:3:1:’1)
Hash(topic2 “Computer Architecture”)	(5, 1:1:2:1:2)
Hash(topic2 “Decrypt Model”)	(5, 1:2:3:1:2)

**Table 3.** Attribute database file.

<b>Attribute numbering code</b>	<b>Attribute value</b>
(3, 1:1:1)	title—Fundamentals of C++
(3, 1:2:1)	title— Cryptography and Network Security
(3, 1:1:1)	author— Nance Lambert
(3, 1:2:1)	author—William Stallings
(5, 1:1:2:1:1)	topic1—Computer History
(5, 1:1:2:1:2)	topic2—Computer Architecture
(5, 1:1:2:2:1)	topic—Basic Concept
(5:1:1:3:1:1)	topic—User Defined Function
(5, 1:1:4:1:1)	topic—Selection Statements
(5, 1:2:2:1:1)	topic—Basic Concept
(5, 1:2:3:1:1)	topic1—Encrypt Model
(5, 1:2:3:1:2)	topic2—Decrypt Model

## 6 Database

Our system uses Berkeley DB [1] with B+ tree structure as the database engine to store and manage XML data. Berkeley DB is embedded in the system, which means DB is supplied as a library that links directly into the application to provide fast, reliable, scalable database support. An application that uses the Berkeley DB should include the library and then create, open and operate a database. The DB handle is the handle for a Berkeley DB database, which may or may not be part of a database environment. The program would include something like the following.

```
import com.sleepycat.db.*;  
  
Db table;  
try {  
    table=new Db(null,0);           //create a new Db  
    table.set_error_stream(System.err); //init table properties  
    table.set_errpx("Berkeley Database");  
    table.set_flags(Db.DB_DUP);  
  
    //create database with B+ tree structure named after Fname  
    table.open(Fname, null, Db.DB_BTREE, Db.DB_CREATE, 0644);  
}  
catch(DbException re) { }
```

Berkeley DB storage and retrieval for the access methods are based on (key, data) pairs. Both key and data items are represented by Db objects. Key and data byte strings may refer to strings of length zero up to essentially unlimited length.

## 7 Query Examples

Based on the previous sample XML data and the indexing strategy, we shall give two examples to show how to efficiently query the specific indexing system with numbering code associated with each node. The query processor of the system implements XPath1.0.

### 7.1 Example 1

This is a simple example: `//unit1/chapter2`

This query is to find all the nodes, chapter2, which are descendants of the node, unit1, in the entire document.

- First, the query is decomposed into two elements, unit1 and chapter2. Meanwhile, chapter2 should be a descendant of unit1.
- Then, the system finds the codes of the element unit1 from the element index file using hash(unit1) as the an entry. Codes of element chapter2 are found in the same way. It turns out that there are two unit1 elements and two chapter2 elements in the XML document. The searching results are:

Hash(unit1) - node A	(3, 1:1:2)
Hash(unit1) - node B	(3, 1:2:2)
Hash(chapter2) - node C	(4, 1:1:2:2)
Hash(chapter2) - node D	(4, 1:2:3:1)

- Third, the system compare each of the unit1 elements and each of the chapter2 elements to determine their ancestor-descendant relationship. It is clear that node *C* is an descendent of node *A*, because  $c(C)=1:1:2:2$  and  $c(A)=1:1:2$  and hence  $c(C)_3 \text{ XOR } c(A) = 0$ . But *C* is not a descendent of *B*. By the same token, *D* is neither a descendent of *A* nor of *B*.

To conclude, for query `//unit1/chapter2` on the sample XML data, we would have a Nodeset returned with only one element, chapter2, with code (4, 1:1:2:2), which is node *C*.

## 7.2 Example 2

This example involves attributes. `//book[@author= "William Stallings"]/@title` In this example, the query is to look for all books, which have an attribute author of value “William Stallings.” Having found this set, return the values of all of the attribute, title, of each selected element, book.

- First, the query is decomposed into one element, book, and two attributes author and title. In addition, the two attributes are children of the same element, book.
- Then, codes of all elements with name book are found from the element index file with hash(book) as an entry. Codes of attributes author are found from the attribute index file with hash(author + “William Stallings”) as the entry. The results are:

Hash(book) - node E	(2, 1:1)
Hash(book) - node F	(2, 1:2)
Hash(author + ‘William Stallings’) - node G (3, 1:2:1)	

- Third, using Lemma 1 we can quickly determine that node F (with code  $c(F)=1:2$ ) is an ancestor of node G (with code  $c(G)=1:2:1$ ) but E and G do not have such relationship. Hence we have obtained the Nodeset containing one node, node F, as the book element satisfying the query. Then we look for the value of the attribute title of the previously selected elements. For this, the code of node *F*, (3, 1:2:1), is used as the index to the attribute database to get all the entries started with title. The searching result would be the following: (3, 1:2:1) *Title— Cryptography and Network Security*.

The final result of the query is returned as an XML document as:

```
<? xml version='1.0' encoding='utf-8'?>
<xml file: "http://www.cps.cmich.edu/~tang1/bookstore.xml->
<Query //book[@author= "William Stallings"]/@title/>
<Result title="Cryptography and Network Security"/>
```

Although the two examples are quite simple, they illustrate the basic operations for search the data for a given query. Decomposition of the query path expression and search for the data using B+ tree indexing are performed just like many other query processing systems. However, the partial search results must satisfy the path condition. That is, they must have the ancestor-descendant relationships. With our coding method, this can be done in constant time.

### 7.3 Other Examples

We have tested several larger XML data files. The characteristics of two of the files are shown in Table 4 below.

**Table 4.** Characteristics of the test data files.

XML file	Data size	Max depth	Avg depth	# Element	# Attribute
Auction data (ubid.xml)	20 KB	5	3.76	300	0
Sigmod data (sigmod.xml)	467 KB	6	5.14107	11526	3737

The Sigmod XML data involves indexing and retrieving attributes that is much slower than performing the same operations on elements. The execution time is listed in the following tables. It is shown in the experiments that the execution time does not increase with longer paths for ubid.xml (Table 5), but increased significantly for Sigmod.xml that involves attributes (Table 6).

**Table 5.** Execution time of queries on the ubid.xml file.

Depth of path	Query path	Indexing time (ms)	Query time (ms)
1	//listing/shipping-info	2051	392
2	//listing//current-bid	1719	353
3	//listing//bidder-name	2013	355

**Table 6.** Execution time of queries on the ubid.xml file.

Depth of path	Query path	Query time (ms)
1	//issue/number	1871
2	//issue//article	49941
3	//issue//title	127827

We are currently testing large XML files of size in the range of several MB to 2GB.

## 8 Conclusion and Future Work

We have developed an index-based XML query system to answer user's queries in the form of path expression. The indexing system uses a new numbering scheme to

efficiently determine the ancestor-descendant relationship between nodes in the XML hierarchy in constant time. This node numbering method not only is efficient in determine the ancestor-descendent relationship but also overcomes the problems other numbering approaches didn't solve, that is, the flexibility for inserting new nodes in XML data without re-numbering the existing nodes.

The XML query system creates index structures for the elements, attributes, and the data at the leaf nodes, and builds the database using the Berkeley DB utilities, particularly the B+ tree library.

Currently we are testing the system on large-size XML data and doing some comparative analysis on the performance with other methods. We also plan to investigate the problem of frequently updated XML documents.

## References

1. Berkeley DB. <http://www.sleepycat.com/docs/ref/refs/bdb-usenix.html>
2. D. Chamberlin, D. Florescu, J. Robie, J. Simon, and M. Stefanescu: XQuery: A Query Language for XML. W3C working Draft, Technical Report WD-xquery-20010215, World Wide Web Consortium (2001)
3. J. Clark and S. DeRose: XML Path Language (XPath) Version 1.0. W3C Recommendation. Technical Report REC-xpath-19991116, World Wide Web Consortium (1999)
4. B. F. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, and M. Shadmon: A Fast Index for Semistructured Data. Proceedings of the 27th International Conference on Very Large Databases, Rome, Italy (2001)
5. P. F. Dietz: Maintaining order in a Linked List. Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, San Francisco, California (1982)
6. D. Egnor and R. Lord: Structured Information Retrieval Using XML.  
<http://www.haifa.il.ibm.com/sigir00-xml/final-papers/Egnor>
7. Emily: <http://www.emilysolutions.com>
8. Etymon: <http://www.etymon.com/amberfish/index.html>
9. H. Katz: XQEngine. <http://www.fatdog.com>
10. D. Kha, M. Yoshikawa, and S. Uemura: An XML Indexing Structure with Relative Region Co-ordinates. Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany (2001)
11. R. R. Larson: Cheshire II Project. <http://cheshire.lib.berkeley.edu>
12. Q. Li and B. Moon: Indexing and Querying XML Data for Regular Path Expressions. Proceedings of the 27th International Conference on Very Large Databases, Rome, Italy (2001)
13. R. Luk, A. Chan, T. S. Dillon, H. V. Leong: A survey of search engines for XML documents. SIGIR 2000 Workshop on XML and Information Retrieval, Athens, Greece (2000) 1-9
14. J. McHugh, J. Widom, S. Abiteboul, Q. Luo, and A. Rajaraman: Indexing Semistructured Data. Technical Report, Stanford University (1998)
15. F. Rizzolo, A. Mendelzon: Indexing XML Data with ToXin. Fourth International Workshop on the Web and Databases (in conjunction with ACM SIGMOD 2001). Santa Barbara, California (2001)
16. J. Shanmugasundaram, et al.: Relational Databases for Querying XML Documents: Limitations and Opportunities. Proc. Of the 25th International Conference on VLDB, Edinburgh, Scotland (1999)
17. Structured Information Manager. <http://www.simdb.com>.

18. F. Tian, D. J. DeWitt, and J. Chen: The Design and Performance Evaluation of Alternative XML Storage Strategies: SIGMOD Record special issue on Data Management Issues in E-Commerce (2002)
19. V. Turau: Making Legacy Data Accessible for XML Applications.  
<http://www.informatik.fh-wiesbaden.de/turau/veroeff.html>.
20. J. Yoon, V. Raghavan, V. Chakilam, and L. Kerschberg: BitCube: A Three-Dimensional Bitmap Indexing for XML Documents. *Journal of Intelligent Information Systems*, **17**, (2001)
21. M. Yoshikawa, T. Amasaga, T. Shimura, and S. Uemura: XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases. *ACM Trans. on Internet Technology* (2001)
22. XDisect - XML Refined Search. <http://www.pybiz.com/products/xdisect>
23. Xset Enabling XML Applications. <http://www.cs.berkeley.edu>

# Improving Software Engineering Practice with HCI Aspects

Xavier Ferre, Natalia Juristo, and Ana M. Moreno

Universidad Politecnica de Madrid

Campus de Montegancedo, 28660 - Boadilla del Monte (Madrid), Spain

{xavier,natalia,ammoreno}@fi.upm.es

<http://www.ls.fi.upm.es/udis/>

**Abstract.** Techniques from the HCI (Human-Computer Interaction) field have been used for the development of usable software products for a long time, but their use is often not integrated with software engineering practices. In this work we describe an approach for bridging the gap between software engineering and HCI, by offering orientation to software practitioners on the application of HCI techniques and activities. For this purpose, we have carried out a survey in HCI literature to define the activities in a user-centered development process, and to select the HCI techniques that are more appropriate for integration into software engineering practice.

## 1 Introduction

According to ISO Standard 9241-Part 11, usability is “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use” [7]. Usability is a critical quality factor for the success of a software product. In this direction, Larman states that there is probably no other technique with greater disproportion between its importance for the success of software development and the lack of attention and a formal education, as usability engineering and the design of the user interface [9]. Nevertheless, a change can be seen in the attention paid to usability. An increasing number of software development companies are beginning to consider usability as strategic for their business, and they are pursuing the aim of integrating HCI practices into their software engineering processes. Some proposals for integration ([1], [14]) present ad-hoc solutions which have been created for particular software development organizations, but they lack a generic approach to be applied to organizations with different characteristics.

Techniques aimed to increase the usability level of the software product are applied following development processes which are particular to the HCI field, and these processes are not formalized from the point of view of software engineering. Therefore, they are not easy to transfer to the formalized software engineering processes. One of the virtues of the HCI field lies in its interdisciplinary nature, but this characteristic is at the same time the greatest obstacle for its integration with software engineering. While the theoretical and practical basis of HCI comes from Sociology, Psychology, Ergonomics, Graphical Design and so on; software engineers have a clear engineering focus. Both fields speak different languages, and they approach software development from a very distinct perspective.

According to Constantine and Lockwood [2], the classic view of software quality has focused primarily on internal efficiency and reliability of the code in operation; but the view of software begins to shift outward from a limited internal perspective to an external one that more fully considers customers and end users. Usability, along with utility and capability, are clearly seen as key factors in software quality. Software development is mostly focused on internals, in processing logic and data organization to fulfill narrow concepts of functional objectives. These aspects of the software system are almost completely alien to the final user of the product. However, the system-user interaction has been traditionally considered as a secondary issue. Despite mentioning as an objective to build a system that satisfies user needs, after establishing a set of requirements, the development effort is mostly carried out without further contact with the final users (at least not before a first “polished” version of the software product is produced). Usability is often identified with just the design of the graphical user interface, a part of the system that is developed at the end of the software development process. This kind of approach is responsible for the development of systems with a very low usability level, whose usability problems are identified once their correction is too costly.

On the contrary, HCI experts study the users and the way they interact with the system from the beginning of the development effort. HCI experts employ a set of techniques for interaction design, and for evaluation of software products with real users.

The aim of the present work is to approach the integration of HCI techniques into the software development process from a software engineering perspective, making possible the application of HCI techniques by software developers (not HCI experts), or at least that software developers may incorporate the ‘caring for usability’ development philosophy present in HCI practices into their development practices. For this purpose, we have begun by studying the characteristics of a user-centered development process (the approach to development taken in HCI) in the HCI literature. Then we have identified the activities in a user-centered development process, and we have finished the HCI literature survey by studying the classification of the main HCI techniques according to a user-centered development scheme. In order to make this scheme understandable by software developers, we have finally mapped the activities in a user-centered process to the usual activities undertaken in a generic software development process, as understood in software engineering. Average developers may use our proposal to decide how and where the HCI techniques may fit with the rest of techniques they usually employ. Our proposal is generally applicable to interactive software development, given that the existing development process is based in iterative development.

The research presented in this work has been carried out as part of the project STATUS, financed by the European Commission (IST - 2001 - 32298). The project includes between its objectives, the output of methodological guidelines for the integration of usability techniques into the software process, which we are presenting here.

## 2 Definition of a User-Centered Development Process

As a first step for the integration of HCI techniques and activities into the software development process, we have carried out an HCI literature survey to identify the

characteristics that a software development process should have for it to be considered user-centered and, therefore, support the development of a final product with a high level of usability. These characteristics may be used by any organization to decide whether its software process can serve as a basis for the integration of usability techniques into software development or, on the contrary, it has to consider migrating to another type of process if it really intends to go for usability.

Albeit strictly in reference to user-centered *design*, Preece et al. gives a definition of user-centered that is potentially of interest for our process requirements search. It should [12]:

- be user-centered and involve users as much as possible so that they can influence the design,
- integrate knowledge and expertise from the different disciplines that contribute to HCI design,
- be highly iterative so that testing can be done to check that the design does indeed meet user requirements.

The ISO Standard 13407 on Human-Centered Design Processes for Interaction Systems [8] defines that the incorporation of a human-centered approach is characterized by the following:

- the active involvement of users and a clear understanding of user and task requirements;
- an appropriate allocation of function between users and technology;
- the iteration of design solutions;
- multi-disciplinary design.

Looking for a different point of view, we find that [2] defines the elements of a usage-centered approach as follows:

- Pragmatic design guidelines.
- Model-driven design process.
- Organized development activities.
- Iterative improvement.
- Measures of quality.

Shneiderman states that a process that supports usability needs to be non-hierarchical in the sense that it is neither strictly top-down nor bottom-up; and it is radically transformational, implying the production of interim solutions that could ultimately play no role in the final design [13]. This leads to an iterative development approach.

From the characteristics of a proper user-centered process detailed above, we can extract three main issues that need to be dealt with: user involvement; adequate understanding of user and task requirements; and iterative development. For the fulfillment of the first two requirements, we detail below a set of HCI techniques that may help to achieve them. The techniques specify when and how the user should be incorporated and what usability knowledge should be applied and when. On the other hand, iterative development is an intrinsic development process requirement. Therefore, according to [12], the organization's design process should be highly iterative to support usability and, consequently, to be able to incorporate the HCI practices.

Iterative development is a must. The usability level of the system cannot be predicted in advance. Some kind of usability evaluation is needed at the end of every iterative cycle. Therefore, the requirement of an iterative process is closely linked to the need to perform quality measures at the end of each cycle, and it is the only requirement to be met by a development process applied by a software development organization, for it be a candidate for the integration of HCI aspects.

The other two requirements: user involvement and adequate understanding of user and his or her tasks, are also changes in developers' way of doing things, although these changes can be accomplished by the application of HCI techniques.

### 3 Representative HCI Activities

The HCI field is diverse, and there is no general agreement on the set of activities that are part of a user-centered development process. For that reason, we have performed a literature survey in order to obtain a set of activities/tasks that lead to the development of software systems with an acceptable usability level.

Although software engineering has made efforts towards software process formalization, HCI authors have not strived for formality. On the contrary, they propose tasks, activities, process heuristics and advice, which are not integrated into a process that can be used as a framework for development. The sources vary as to the extent of formalization. The set of usability-related activities proposed in the HCI field are detailed in Table 1, where sources follow an order of increasing formalization from left to right. We have analyzed the activities proposed by the different authors in order to extract the common ones or, at least, the activities that are at the same abstraction level and are common to several sources. Our aim is to be able to easily compare the different proposals, and, therefore, we have grouped activities that refer to the same concept in the same row. Each row has been labeled (first column in the table) with the most general term or the term more often used by the authors studied, and there is one column per author that contains the respective activity that they propose. Where the author packs several tasks into the same activity, the complete name given for the activity (for example, Systems/ Tasks / Functional / User Analysis) has been included in the table. Some authors describe a generic activity that includes the activity we are considering as a subtask. In these cases, the specific subtask is highlighted in italics. On the other hand, where the author proposes several activities that match one of our activities, they are listed using an asterisk (\*). For activities not mentioned in the source, the cell contains a dash ('-').

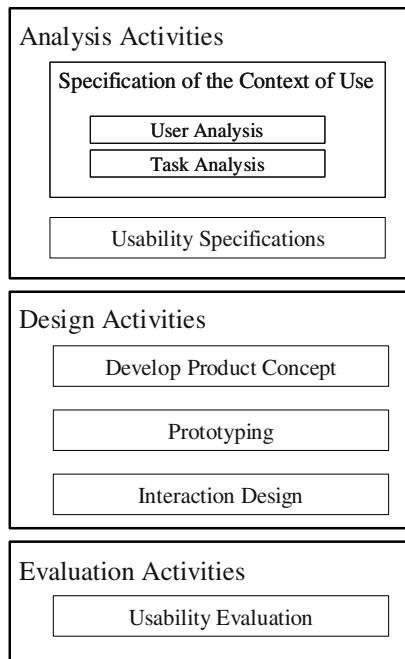
There is a clear trend in most of the sources considered as regards the activities to be done: usability specifications, prototyping and usability evaluation. The specification of the context of use, either as a complete analysis of a variety of user and organizational issues or just with an aim of knowing the user, is also quite prevalent among the different authors. We have found more discrepancies, and less information on other design activities, like the development of the product concept and the interaction design. While some authors give no clues as to the design activity, apart from labeling it as user-centered or advocating iterative design, Constantine and Lockwood [2] are more specific with respect design issues, criticizing the trend in usability engineering that focuses almost exclusively on usability testing.

**Table 1.** Usability Activities by Source.

Activity	Nielsen93	ISO99	Shneiderman 98	Hix93	Preece94	Wixon 97	Constantine 99
SPECIFICATION OF THE CONTEXT OF USE	Know the user	Understand and specify the context of use	Perform research and needs analysis	Systems/ tasks / functional / user analysis	Task analysis / functional analysis	* Specify and categorize the users * Conduct a task analysis	Task modeling
USABILITY SPECIFICATIONS	Goal Settings	Specify the user and organizational requirements	Design concepts and key-screen prototype ( <i>create specific usability objectives based on user needs</i> )	Requirements / Usability Specifications	Requirements specification	* Define quantitative usability goals * Set levels of desired usability for each goal	-
DEVELOP PRODUCT CONCEPT	-	-	Develop product concept	Conceptual design	Conceptual design / formal design	-	-
PROTO-TYPING	Proto-typing	Produce design solutions ( <i>make design solutions more concrete using simulations, models, mock-ups, etc.</i> )	Design concepts and key-screen prototype	Rapid prototyping	Prototyping	-	-
INTERACTION DESIGN	Iterative Design	Produce design solutions	Do iterative design and refinement	Design & design representation	Conceptual design / formal design	-	Interface content modeling
USABILITY EVALUATION	Interface Evaluation	Evaluate design against requirements	Do iterative design and refinement ( <i>conduct full-scale usability tests</i> )	Usability evaluation	Evaluation	Test the product against usability goals	Usability inspection

The resulting usability activities (the left column in Table 1) are represented in Fig. 1, grouped according to the generic kind of activity to which they belong: Analysis, design or evaluation. Note that the Specification of the Context of Use is decomposed in User and Task Analysis. Some authors ([5], [15]) differentiate between both activities, even if they recognize that they are closely related. We have chosen the terminology of the Standard ISO-13407 [8] because it better reflects the close relationship between both subactivities.

Prototypes are widely used in other fields than HCI, in particular related to iterative development, but what HCI may offer is the particular usage of prototyping in order



**Fig. 1.** Activities in a User-Centered Development Process.

to get greater degrees of user involvement, and to consider alternative designs. The most useful prototypes for this purpose are the less elaborate ones, such as paper prototypes.

Apart from Prototyping, the other two design activities identified have no such wide presence in HCI literature. The development of the product concept is based on mental models ([11], [12]): When the product concept is vague, ambiguous, inconsistent or obscure, there will be a divergence between the user mental model of the system and the design model that developers work with. The activity of Develop the Product Concept is not alien at all to software engineering; it is a modeling effort aimed to ensure a proper communication between the members of the development team. The pursuit of consistence and logic in a design is a clear engineering endeavor, and we just need to add the finality of producing a product concept that matches the expectations and needs of the final user.

Interaction Design is an activity that is not defined in great detail, and its definition noticeably varies between different authors. Interaction design and the design of the user interface are closely related, but they are different [3]. Some authors refer to "User Interface Design" ([10], [5], [13]), while others use the term "Interaction Design" ([12]) or just "Design" ([15], [8]). Additionally, Constantine and Lockwood refer to "Dialogue Design" or "Visual Design" [2]. As the design of the interaction is critical for the usability level of the final product, we have incorporated Interaction Design as an activity in the set of activities in a user-centered development process in Fig. 1.

**Table 2.** Analysis-Related Techniques.

Analysis-Related Techniques		Nielsen93	Preece94	Hix93	Shneiderman98	Constantine99
Specification of the Context of Use	Functional Analysis (5)	Functional Analysis		Functional Analysis		
	Needs Analysis (5) (8)			Needs Analysis		
	Competitive Analysis	Competitive Analysis				
	Financial Impact Analysis (7) (8)	Financial Impact Analysis				
	Contextual Inquiry		Contextual Inquiry	Contextual Inquiry		
	Ethnographic Observation		Ethnography		Ethnographic Observation	
	Sociotechnical Approach (1) (8)		Sociotechnical Approach			
	User Analysis	Structured User Role Model				Structured User Role Model
		User Profiles (4)	Individual User Characteristics		User Profiles	Usage Profiles
		Operational Modeling				Operational Modeling
Task Analysis	Essential Use Cases					Essential Use Cases
		HTA (4)		HTA		
	Cognitive Task Analysis		Cognitive Task Analysis			
		GOMS	GOMS	GOMS	GOMS	
		TAG (4)	TAG		TAG	
	Object-action Interface Model (4)				Object-action Interface Model	
		Scenarios	Scenarios		Scenario Development	
Usability Specifications	Based on Benchmark Tasks		Benchmark Tasks	Benchmark Tasks		
	Based on Preference Questionnaires			User Questionnaires		

## 4 Selection of HCI Techniques

After obtaining a characteristic set of HCI activities, we have continued our literature survey by focusing on the set of techniques commonly applied in the HCI field. The same sources as for the HCI activities survey were chosen, except for [15] and [8]. We have not considered these two sources, because they focus on activities in a usability-oriented process and give few details on the individual techniques to be used.

**Table 3.** Design-Related Techniques.

Design-Related Techniques			Nielsen93	Preece94	Hix93	Schneiderman98	Constantine99
Develop Product Concept	Conceptual Design				Conceptual Design		
	Post-It Notes					Post-It Notes	
	JEM					JEM	
	Visual Brainstorming			Visual Brainstorming			
Prototyping	Prototyping Strategies (6)	Rapid Prototyping		Rapid Prototyping	Rapid Prototyping		
		Incremental Prototyping		Incremental Prototyping			
		Evolutionary Prototyping		Evolutionary Prototyping			
	Kinds of Prototypes	Requirements Animation	Mock-ups (Limited Impl.)				Active Prototypes
		Non-functional Prototypes	Chaffeuried Prototypes	Chaffeuried Prototyping			
		Paper Prototypes					Passive Prototypes
		Wizard of Oz		Wizard of Oz			
Interaction Design	Screen Pictures				Screen Pictures		
	Use Cases					Use Cases	
	Grammars (1)					Grammars	
	Menu-Selection and Dialog Box Trees					Menu-Selection and Dialog Box Trees	
	Context Navigation Map				Interface State Transition Diagrams	Transition Diagrams and State-charts	Context Navigation Map
	UAN (4)				UAN	UAN	
	Other Design Techniques	<i>Both-And Design</i>					<i>Both-And Design</i>
		Design Alternatives Management	<i>Parallel Design</i>	<i>Parallel Design</i>			
		Impact Analysis	Impact Analysis	Impact Analysis	Cost / Importance Analysis		
		Help Design	Organizing Help by Use Cases				Organizing Help by Use Cases
		Design Rationale (5)	IBIS, PHI, Design Space Analysis, Claims Analysis		IBIS, PHI, Design Space Analysis, Claims Analysis		

**Table 4.** Usability Evaluation Techniques for Usability Testing.

Usability Testing Techniques		Nielsen93	Preece94	Hix93	Shneiderman98	Constantine 99
Usability Tests	Thinking Aloud	Thinking Aloud	Think Aloud Protocol	Concurrent Verbal Protocol Taking		Talk to Me (think out loud)
	Constructive Interaction	Constructive Interaction				
	Retrospective Testing	Retrospective Testing	Post-Event Protocol	Retrospective Verbal Protocol Taking		
	Critical Incident Taking			Critical Incident Taking		
	Coaching Method	Coaching Method				
	Measured Performance					Measured Performance
	Post-Test Feedback					Post-Test Feedback
	Laboratory Usability Testing				Usability Testing and Laboratories	Laboratory Usability Testing
	Field Usability Testing	Direct Observation	Usability Assessment through Observation	Direct Observation		
		Beta-Testing				Beta-Testing
		Indirect Observation	Video Recording	Video Recording		
		Verbal Protocol		Verbal Protocol	Audio-taping	

We have experienced the same difficulty than in the previous section. The HCI field is very heterogeneous, and we have found a great diversity of techniques. After merging the techniques suggested by different authors that refer to the same basic technique, we still have eighty-two techniques. This is an excessive number of techniques to provide to software developers, especially given that some of them are redundant and a certain number have almost no interest for general software development projects. For the purpose of selecting the techniques more appropriate for integration into software engineering practice, the techniques have been divided into groups, according to a classification of the main kind of activity. We have represented the techniques in several tables: One table for analysis-related techniques (Table 2),

one table for design-related techniques (Table 3), and four tables for evaluation-related techniques (Tables 4 to 7). Each table contains a column for each author, and the left-hand columns specify the technique category and chosen name. For each technique, we have chosen the name we consider to be the most representative. Techniques that are in the same row refer to the same basic technique. The possibility of variants of some techniques is also considered; and then the general technique and each of the variants have their corresponding rows (for example, Table 4 shows the Thinking Aloud technique along with four of its variants). The tables also show the selected as candidate techniques. Techniques on a white background are selected as candidates for inclusion in the software development process. Techniques that have not been selected appear on a grey background, and they have between brackets an indicator of the reason for not being selected, as detailed in Table 8. A more detailed description of the selection process can be found in [4].

A technique is discarded due to one or more of the reasons in Table 8. The techniques appearing in italics have been selected, albeit for optional application when the project meets certain characteristics. Analysis techniques are summarized in Table 2, and design techniques in Table 3.

Evaluation techniques are summarized in four tables: Table 4 presents techniques for usability testing, Table 5 for expert reviews, Table 6 for follow-up studies of installed systems and, finally, Table 7 summarizes the rest of usability evaluation techniques.

**Table 5.** Usability Evaluation Techniques for Expert Reviews.

Expert Review Techniques			Nielsen93	Preece94	Hix93	Shneiderman98	Constantine99
Expert Reviews	Heuristic Evaluation		Heuristic Evaluation	Heuristic Evaluation	Heuristic Evaluation	Heuristic Evaluation	Heuristic Evaluation
	Inspections	Conformance Inspections		Standards Inspection		Guidelines Review	Conformance Inspections
		Consistency Inspection		Consistency Inspection		Consistency Inspection	Consistency Inspection
		Collaborative Usability Inspections					Collaborative Usability Inspections
	Walk-throughs	Pluralistic Walk-through	Pluralistic Walk-through	Pluralistic Walk-through			Pluralistic Usability Walk-through
		Cognitive Walk-through		Cognitive Walk-through		Cognitive Walk-through	Cognitive Walk-through

**Table 6.** Usability Evaluation Techniques for Follow-up Studies of Installed Systems.

Techniques for Follow-Up Studies		Nielsen93	Preece94	Hix93	Shneiderman98	Constantine99
Follow-up Studies of Installed Systems	Questionnaires	Questionnaires and Interviews	Questionnaires and Surveys			
	Interviews	Questionnaires and Interviews	Interviews		Interviews and Focus Group Discussions	
	Structured Interviews		Structured Interviews	Structured Interviews		
	Flexible Interviews		Flexible Interviews			
	Focus Groups	Focus Groups			Interviews and Focus Group Discussions	
	Logging Actual Use	Logging Actual Use	Software Logging	Internal Instrumentation of the Interface	Continuous User Performance Data Logging	
	Time-Stamped Key presses		Time-Stamped Key presses			
	Interaction Logging		Interaction Logging			
	User Feedback	User Feedback			Online Suggestion Box or Trouble Reporting	
	Online or Telephone Consultants				Online or Telephone Consultants	
	Online Bulletin Board or Newsgroups				Online Bulletin Board or Newsgroups	
	User Newsletters and Conferences				User Newsletters and Conferences	
Surveys			Questionnaires and Surveys		Surveys	

**Table 7.** Other Usability Evaluation Techniques.

Other Usability Evaluation Techniques	Nielsen93	Preece94	Hix93	Shneiderman98	Constantine99
Experimental Tests (2)		Traditional Experiments		Controlled Psychologically Oriented Experiments	
Predictive Metrics		Analytic Evaluation Methods			Usability Assessment Based on Predictive Metrics
Acceptance Tests (5)				Acceptance Tests	
Cooperative Evaluation (2) (8)		Cooperative Evaluation			
Participative Evaluation (1)		Participative Evaluation			

**Table 8.** Reasons for Discarding a Technique.

1	It is a special technique for projects with specific characteristics, so it is not generally applicable
2	It is alien to software engineering, so developers will find it very difficult to learn to use it
3	Its application will require the use of extra resources from outside the project team
4	It is made redundant by another selected technique. That is, the expected benefits provided by the application of the technique are already covered by a selected technique, and the selected technique offers some additional advantages
5	It is not specifically an HCI technique, so it does not make sense to include it in an HCI addition to the development process
6	It deals with development process issues, and there are other reasons apart from usability to be taken into account. It must be dealt with in the context of the whole development process
7	The technique is directed at gaining management support for usability activities in the development process. We are working with the hypothesis of an organization that is already willing to incorporate HCI practices, so this kind of support is pre-requisite for the usage of our proposal
8	It is presented by just one author, and we consider that it is not generally accepted as a usability technique in the field. This reason will be considered only in conjunction with other reasons, never by itself

## 5 Fitting HCI Activities into Mainstream Development

If we want that software developers use the HCI techniques that we have compiled, we need them to be expressed according to terminology and concepts that are familiar to developers. The scheme of activities and techniques we have obtained from the HCI field is not very useful for an average developer, because it is based on HCI concepts and terminology, which are somehow alien to them. Therefore, we need to adapt the activity scheme (and also the techniques) from HCI to the activities of a generic software development process.

For the definition of the set of activities in a generic development process, we have mainly based on the SWEBOK (SoftWare Engineering Body Of Knowledge) [6] developed by the IEEE Computing Society.

Table 9 shows the relationship between the activities in a user-centered development process and the activities in a generic development process.

**Table 9.** Mapping between Development Activities Affected by Usability and HCI Activities.

Development Activities Affected by Usability			Activities in a User-Centered Development Process
Analysis (Requirements Engineering)	Req. Elicitation		Specification of the Context of Use - User Analysis
	Req. Analysis	Develop Product Concept	Develop Product Concept
		Problem Understanding	Specification of the Context of Use - Task Analysis
		Modeling for Specification of the Context of Use	Prototyping
	Req. Specification		Specification of the Context of Use - User Analysis
	Req. Validation		Usability Specifications
Design	Interaction Design		Walkthroughs (Usability Evaluation - Expert Evaluation)
Evaluation	Usability Evaluation	Usability Testing	Usability Evaluation - Usability Testing
		Expert Evaluation	Usability Evaluation - Expert Evaluation
		Follow-Up Studies of Installed Systems	Usability Evaluation - Follow-Up Studies of Installed Systems

Regarding analysis, usability activities are intermingled with other analysis activities, so we will integrate HCI activities in analysis with the activities in a generic development process (as indicated in the SWEBOK) with which they are more closely related. There are two activities considered as design activities in HCI, but considered as analysis in software engineering: Prototyping is traditionally used in software engineering for the task of Problem Understanding, while the Development of the Product Concept is a kind of design known as innovative design, which is usually undertaken as part of Requirements Engineering. The SWEBOK does not consider innovative design as part of software design, but as part of requirements analysis efforts. In addition to that, we have that Walkthroughs are a kind of usability evaluation that can be used for the validation of the products of analysis, and that is the reason why it is mentioned in the mapping in Table 9.

Unlike analysis, we have in design and evaluation that HCI activities are quite independent from the rest of development activities, so we have added new activities to accommodate them in the scheme. Regarding design, we have defined a new development activity called Interaction Design.

Usability evaluation is also performed independently from the rest of evaluation activities, and for that reason we have defined a Usability Evaluation activity in the generic scheme. Usability evaluation has a high level of complexity due to the diversity of existing HCI techniques for that purpose, so we have decomposed this activity into the three main kinds of usability evaluation activities: Usability Testing, Expert Evaluation, and Follow-Up Studies of Installed Systems.

## 6 Conclusions

We have presented a strategy for the introduction of HCI techniques and activities into mainstream software engineering practice, by presenting HCI techniques and activities organized according to a scheme that may be mapped to the activities being carried out in any software development organization.

The main reasons for the current not integration of both disciplines (HCI and software engineering) lie in their different terminology and approach to software development. We propose to use software engineering processes and terminology to characterize HCI aspects, so they can be assimilated by average software developers.

For a software development organization wanting to improve the usability of their software products, it is very appealing to have a set of HCI techniques to be incorporated to their current software development process, without need for them to change it for a completely new process. The existing in-house development process does not need to be abandoned, except in the case that it is not iterative. Given the current trend towards iterative development, we consider that this only requirement for the existing software development process is not too restrictive.

For the elaboration of our proposal, we have collaborated with the two software development companies that are part of the STATUS project consortium. They work in the e-commerce domain, where usability is specially critical. After finishing the work presented in this article, we have offered the resulting scheme to both companies and we have received an encouraging initial response from them. With the feedback they provide after using the scheme in two real projects, we plan to refine the scheme to expand it in the direction(s) where developers need more guidance.

## References

1. Anderson, J., Fleek F., Garrity, K., Drake. F.: Integrating Usability Techniques into Software Development. *IEEE Software*. vol.18, no.1. (January/February 2001) 46-53
2. Constantine, L. L., and Lockwood, L. A. D.: *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley, New York, NY (1999)
3. Ferre, X., Juristo, N., Windl, H., Constantine, L.: Usability Basics for Software Developers. *IEEE Software*. vol.18, no.1. (January/February 2001) 22-29
4. Ferre X., Juristo, N., Moreno. A.M.: STATUS Project. Deliverable D.5.1. Selection of the Software Process and the Usability Techniques for Consideration (2002)  
<http://www.ls.fi.upm.es/status/results/deliverables.html>

5. Hix, D., and Hartson, H.R.: *Developing User Interfaces: Ensuring Usability Through Product and Process*. John Wiley and Sons, New York NY (1993)
6. IEEE Software Engineering Coordinating Committee: *Guide to the Software Engineering Body of Knowledge - Trial Version 1.00*. IEEE Computer Society, Los Alamitos, California (May 2001)
7. ISO: ISO 9241-11. *Ergonomic Requirements for Office Work with Visual Display Terminals*. ISO, Geneva, Switzerland (1999)
8. ISO: International Standard: *Human-Centered Design Processes for Interactive Systems*, ISO Standard 13407: 1999. ISO, Geneva, Switzerland (1999)
9. Larman, C.: *UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process* (2nd. Edition). Prentice Hall PTR (2001)
10. Nielsen, J.: *Usability Engineering*. AP Professional, Boston, MA (1993)
11. Norman, D. A.: *The Design of Everyday Things*. Doubleday (1990)
12. Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., Carey. T.: *Human-Computer Interaction*. Addison Wesley, Harlow, England (1994)
13. Shneiderman. B.: *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, Reading, MA (1998)
14. Radle, K., Young, S.: *Partnering Usability with Development: How Three Organizations Succeeded*. IEEE Software. vol.18, no.1. (January/February 2001) 38-45
15. Wixon, D., Wilson. C.: *The Usability Engineering Framework for Product Design and Evaluation* in Helander. M.G. et al. (eds.): *Handbook of Human-Computer Interaction*. Elsevier North-Holland (1997) 653-688

# Reliability Assurance in Development Process for TOE on the Common Criteria

Haeng-Kon Kim<sup>1</sup>, Tai-Hoon Kim<sup>2</sup>, and Jae-Sung Kim<sup>2</sup>

<sup>1</sup> Department of Computer Information & Communication Engineering  
Catholic University of Daegu, Kyungsan, Kyungbuk, 712-702, South Korea

<sup>2</sup> IT Security Evaluation & Certification Authority  
Korea Information Security Agency, Seoul  
138-803, South Korea

**Abstract.** Security begins with good software code and high quality testing of the code, and it continues with the process used to identify corrected and patch security vulnerabilities and with their auditing based on recognized standards. Security is an important aspect of software systems, especially for distributed security-sensitive systems. The Common Criteria (CC) is the standard requirements catalogue for the evaluation of security critical systems. Using the CC, a large number of security requirements on the system itself and on the system development can be defined. However, the CC does not give methodological process support. In this paper, we show how integrate security aspects into the software engineering process. In addition, we also introduce our work on ensuring the reliability assurance in development process for Network Management System as TOE. The activities and documents from the Common Criteria are tightly intertwined with the system development, which improves the quality of the developed system and reduces the additional cost and effort due to high security requirements.

For modeling and verification of critical parts of CBD(Component Based Development) system, we use formal description techniques and model checker, which increases both the understanding of the system specification and the system's reliability. We demonstrate our ideas by means of a case study, the CBD-NMS project.

**Keywords:** Target of evaluation, Security Engineering, Common Criteria, Development Process, Software Engineering, Requirements Engineering, Component Based Development

## 1 Introduction

Developing security critical system is very difficult. Security cannot simply be tested, but has to be ensured during the whole development process. To solve this problem, there are highly sophisticated collections of evaluation criteria that security critical systems have to meet, like the ITSEC security evaluation criteria or their recent successor, the Common Criteria (CC) [1,2,3]. The Common Criteria describes security

related functionality to be included into a system, like authentication, secrecy or auditing, and assurance requirements on its development. The assurance requirements are structured into levels (evaluation assurance levels, EAL). The strictest level is EAL7, where a formal representation of the high-level design and formal proofs of correspondence with the security requirements must be provided. When a system could be evaluated successfully based on the Common Criteria, it is given a certificate, which shows the trustworthiness of the system and is in some cases legally required. However, the CC requirements have complex dependencies and the CC does not give any guidance on how to fulfill them during the development process. Conversely, current software development methodologies such as the V-Model or the Catalysis approach lack adequate support [4,5].

Integration of security into system development is necessary to build secure systems. In this paper, we show how to develop secure systems based on the Common Criteria. For this purpose, the required activities and documents are integrated tightly into the development process. Our methodology is independent of a particular assurance level, and to achieve very high assurance provides the possibility to use formal models for the critical parts of the system. The security measures are carried out during the entire process, as early as possible when they become relevant. This ensures that security problems are discovered when they are still easy to counter. Besides, development documents can be re-used for evaluation purposes and vice versa, and documentation is written when knowledge about the relevant details is still available. The process we used improves the quality (by its requirements on designs, testing or configuration management) and trustworthiness of the system and reduces evaluation time and cost. We demonstrate our ideas by means of the CBD-NMS (Component Based Development-Network Management System). Formal modeling and verification techniques were used to ensure the security of the applied communication protocols (required for evaluation at EAL7). The smooth project realization, the short development time and the high quality of the network management system shows the value of the CC-based process for the development of secure systems.

## 2 An Approach to Security Characterization

Security is an important aspect of software systems, especially for distributed security-sensitive systems. When we assemble systems from existing components, it is vital that we must be clear about the security characteristics of these components and their impact on the target systems. In order to provide such security-related information for components and component-based systems, a model for their security characterization is required to augment our frame-work for component interface definition.

### 2.1 Security of Information Technology Products and Systems

Over the years, there has been much effort in evaluating Information Technology (IT) security, i.e., the security properties of IT products and systems, including hardware, software and firmware. There have been the *Trusted Computer System Evaluation*

*Criteria (TCSEC)* developed in the United States, the *Information Technology Security Evaluation Criteria (ITSEC)* developed by the European Commission, the *Canadian Trusted Computer Product Evaluation Criteria (CTCPEC)*, and the *Federal Criteria for Information Technology Security (FC)* from the United States. Since the early 1990s, the sponsoring organizations of above standards under the coordination of ISO have been working together to align their criteria and create a single international standard of IT security evaluation criteria for general use in the global IT market. The result is the current ISO/IEC International Standard 15408, Common Criteria for Information Technology Security Evaluation, version 2.1—commonly referenced as the Common Criteria or simply CC, approved in December 1999 [1,2,3]. Given the ever-increasing wide-spread use and trade of IT products, IT security concerns are not only for the highly security-sensitive IT products. In fact, any IT products acquired from the market place present certain security risks, although with different levels of sensitivity. To use the acquired IT precuts with confidence, their security properties must be measured and made explicit. The common Criteria represent a coordinated effort addressing this issue.

## 2.2 Overview of the Common Criteria

The part of an IT system or product which should be evaluated based on the CC is called Target of Evaluation (TOE) and has to fulfill different security requirements which are verified by an evaluation authority. The security requirements of the CC are divided into security functional requirements (requirements on the product) and security assurance requirements (requirements on the process) and are structured into classes. The functional requirements are realized in the functions of the system in order to achieve the security objectives of the TOE. The assurance requirements contain measures to be undertaken during development in order to keep the risk for weak points low. They are necessary for guaranteeing the confidence that the TOE meets its security objectives. The number and strictness of the assurance requirements to be fulfilled depends on the Evaluation Assurance Level (EAL), as in table 1, one has chosen for the TOE[1,2,3]. The sub clauses in table1 provide definition of EALs highlighting differences between the specific requirements and the pros characterization of those requirements using bold type. A life-cycle model provides for the necessary control over the development and maintenance of the TOE, if the developer can supply information that shows that the model appropriately minimizes the danger of security violations in the TOE. Information given in the ST about the intended environment of the TOE and about the TOE's security objectives may be useful in defining the model for the portion of the life-cycle after the delivery of the TOE.

## 2.3 Integration of the CC Requirements into the Development

To develop security critical systems based on the CC, we assign the CC activities and products to software development phases. The methodology we use is described on an abstract level, independently from the chosen EAL. It is based on an iterative proc-

**Table 1.** Evaluation assurance level in CC.

Assurance Class	Assurance Family	Assurance Components by Evaluation Assurance Level						
		EAL 1	EAL 2	EAL 3	EAL 4	EAL 5	EAL 6	EAL 7
Class ACM: Configuration management	ACM_AUT				1	1	2	2
	ACM_CAP	1	2	3	4	4	5	5
	ACM_SCP			1	2	3	3	3
Class ADO: Delivery and operation	ADO_DEL		1	1	2	2	2	3
	ADO_IGS	1	1	1	1	1	1	1
Class ADV: development	ADV_FSP	1	1	1	2	3	3	4
	ADV_HLD		1	2	2	3	4	5
	ADV_IMP				1	2	3	3
	ADV_INT					1	2	3
	ADV_LLD				1	1	2	2
	ADV_RCR	1	1	1	1	2	2	3
	ADV_SPM				1	3	3	3
Class AGD: Guidance documents	AGD_ADM	1	1	1	1	1	1	1
	AGD_USR	1	1	1	1	1	1	1
Class ALC: Life cycle support	ALC_DVS			1	1	1	2	2
	ALC_FLR							
	ALC_LCD				1	2	2	3
	ALC_TAT				1	2	3	3
Class ATE: Tests	ATE_COV		1	2	2	2	3	3
	ATE_DPT			1	1	2	2	3
	ATE_FUN		1	1	1	1	2	2
	ATE_IND	1	2	2	2	2	2	3
Class AVA: Vulnerability assessment	AVA_CCA					1	2	2
	AVA_MSU			1	2	2	3	3
	AVA_SOF		1	1	1	1	1	1
	AVA_VLA		1	1	2	3	4	4

process. A single iteration is divided into planning phase, analysis phase, design phase, implementation phase and delivery and operation phase. In every process iteration, a fully functional system is built satisfying a defined set of requirements. After one iteration is finished, the resulting product is ready for evaluation. The following iteration starts again with the planning phase. Now, new requirements in addition to the requirements of the last iteration are defined and at the end of this iteration a new extended system can be evaluated. Fig.1 shows the relations between the CC activities and the development phases. The numbers in columns indicate number of components that are required independently of the chosen EAL and gray bars activities that are not necessary at the EALs. The following sections describe the activities and products for every phase. For the assignment of the CC-products to the phases we decided to consider the requirements of the CC as soon as possible, aiming to detect failures very early in order to counteract early and to reduce project costs[6].

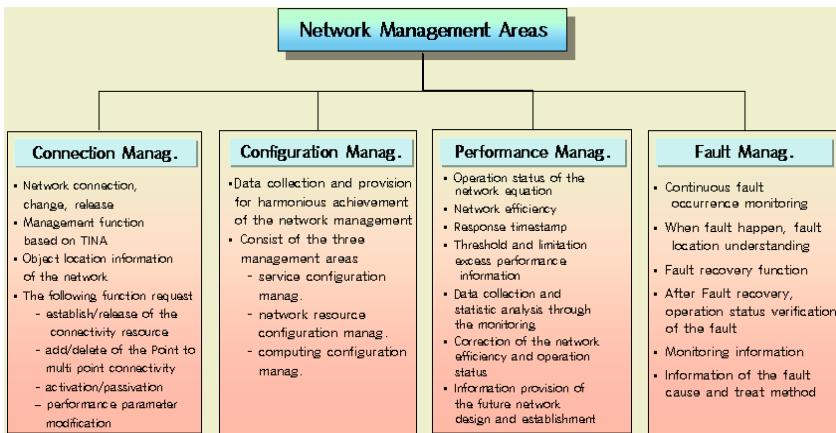


Fig. 1. Network Management Area Information.

### 3 Assurance Developing Process for CBD-NMS on CC

As an application of the CC-based development methodology, we conducted a case study the CBD-NMS project. The aim of CBD-NMS is to develop a secure network management system. CBD-NMS was launched as ITRC project in Korea. During the CBD-NMS project, one iteration of the CC-based process from the planning phase until delivery and operation was passed through. At each time, the team had to fulfill the requirements of the process and to create the corresponding documents. In the following subsections, we describe the CBD-NMS system and how we proceeded during its development, and report the experiences we gained during the project about CC-based development and the CC itself. All documentation on the CBD-NMS project and a demonstration version can be used to provide as a standard assurance in development process for TOE on the Common Criteria from the project of KISA(Korea Information Security Agency)[7].

#### 3.1 The CBD-NMS System

The network management information of the CBD-NMS system is depicted in figure 1. Network management system is a collection of tools for network monitoring and controlling. It involves a single operator interface with a powerful and user-friendly set of commands for performing most or all network management task. As general management checklist, we accept four areas of six managements functional areas based on architecture model of OSI system.

#### 3.2 System Architecture

TINA architecture supports a communication network and a variety of multimedia service in distributed processing environment. It introduces the basic structure providing information of monitoring and analyzing by using the Java-CORBA inter-

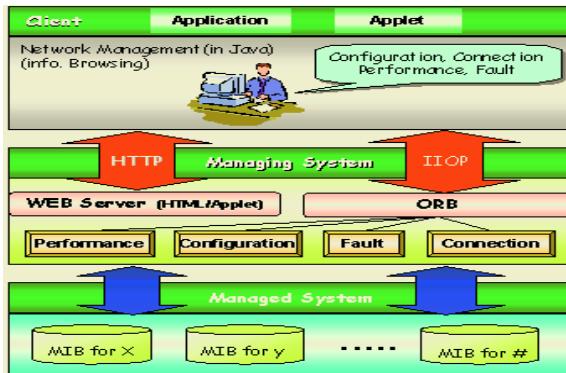


Fig. 2. Network Management System Architecture.

face. It is integrated network management object and service management object. Figure 2 shows the standard architecture of integrated network management system based on Java-CORBA. It provides the inter-operability between applications on different machines as heterogeneous distributed environments manner by using the CORBA. It dynamically monitors and access the network state and information through the user interface system on Java environment [9]. Recently, network management applications are tried to web browser and standard for NM Java interface. The NM applications are required the approach on internetworking through transparent acquisition of network management information. It uses the CORBA as abstraction adapter between network resources. Distributed objects providing management services are abstract and reusable element. Therefore, it can be used as process of CBD directly. In heterogeneous environment, for example, CORBA provide interoperability through dynamic bridging. Also, it is play a role of gateway for integrated network management. As the component composing management system, pattern is generalized and defined as standard idiom for independent selection and management service. It is adapted to dynamical and distributed, exchangeable network management service implementation that provides design relations and template of implementation.

### 3.3 Component Architecture and Network Component Layer

We defined component reference architecture model named ABCD architecture to preceding work for software process definition based on component. Figure 3 shows network management component layer that is classified the components for network management at configuration layer using ABCD architectures[8]. The components have specified network management and located on physical level platform for multi-vendor or applications construction in the distributed computing environment. It also has the integrated API for distributed computing and the components of the existing distributed object services. The common components are necessary to accompany the lower functionality of the all applications. It can be identified component into reliability and reusability through the approach.

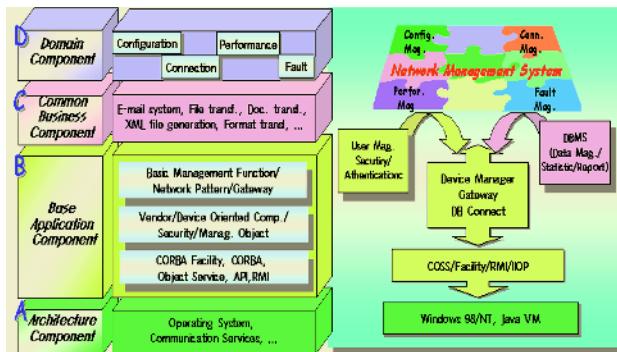


Fig. 3. Network Component Layer.

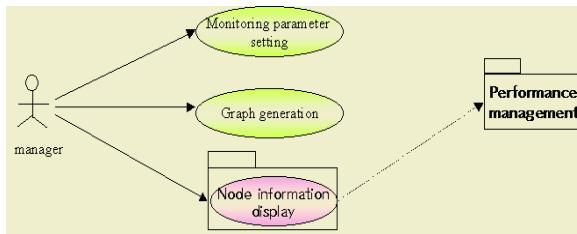


Fig. 4. Use case Diagram of Network Monitoring.

### 3.4 Planning

The development of the CBD-NMS system started with the planning phase. To make the idea more concrete, we prepared a requirements definition without focusing on system details. The requirements definition describes the main **Use Cases** of the CBD-NMS monitoring system as in figure 4. In addition, we defined the system boundary of the CBD-NMS system. During the requirements definition, it was decided to develop the system according to Evaluation Assurance Level 2. After a feasibility study, the decision of developing the CBD-NMS system was made. The next development step was working out the project plan. One group was responsible for consulting regarding the CC evaluation. The second group specialized in NMS issues and system design. The third group was working in formal modeling and verification, and the forth group was responsible for technical aspects and implementation. Creating a configuration management plan was the first activity in this project that is explicitly required to fulfill the CC at EAL2 (class ACM). The configuration management must allow a unique identification of every version of the TOE (system) including all development products divided into configuration items.

A hierarchical version control was needed to maintain the versions of the whole system and its configuration items. The configuration management plan of the CBD-NMS project is describing the single configuration items, the used methods for configuration management and the tool usage. We used the configuration management

tool and the hierarchical version control was performed semi-automatically. Writing the first CC conformable document was difficult. The definitions of the CC terminology are often not clear or even missing.

**Table 2.** Threats to Security.

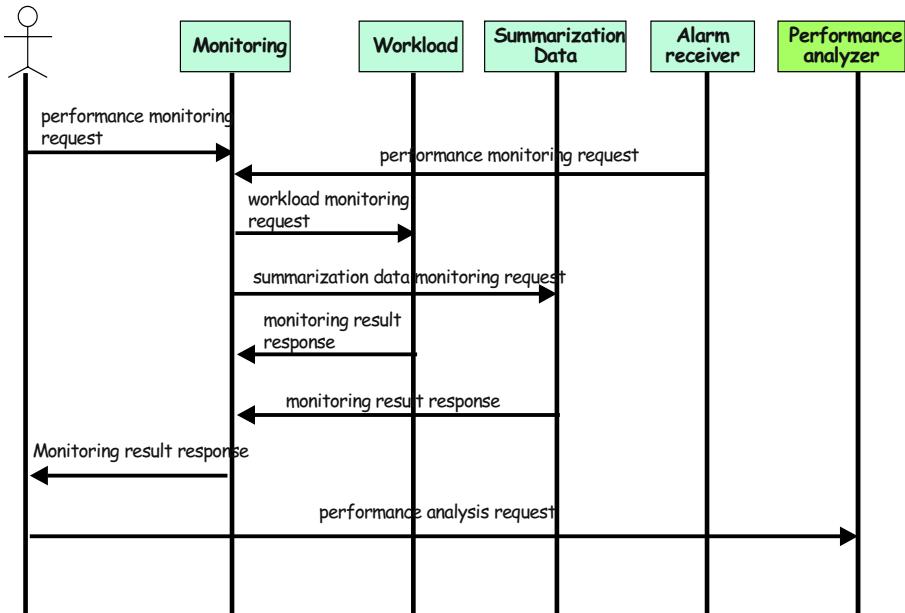
	Threat Name	Threat Description
1	T.ACCESS	An authorized user may be able to access or modify sensitive data for which that user is not authorized to access.
2	T.ACCESS_UNAUTH	An unauthorized person may be able to gain access to the system and to data.
3	T.BYPASS	An attacker may be able to bypass the TOE security functionality and gain unauthorized access to keys, data, and operations.
4	T.CRYPTO_OP	Cryptographic algorithms and/or cryptographic key sizes may be insufficient, allowing them to be deciphered by users that are not authorized access to the encrypted data.
5	T.CRYPTO_MGT	Incorrect cryptographic key destruction may cause an inadvertent disclosure of sensitive information.
6	T.DIG_SIG	Incorrect use of the digital signature and decryption capabilities of the chip could cause a loss of sensitive data, e.g., if a user application improperly supplied a public key to the chip to use in signing when a private key was required or vice versa.
7	T.IMPLEMENT	The TOE may not adequately protect cryptographic keys and data stored in the TOE, causing loss of confidentiality and integrity.
8	T.TAMPER	An attacker may be able to tamper with TSF data or programs.

### 3.5 Analysis

During the analysis phase, the system requirements are defined in detail. Based on the requirements definition, a detailed requirements specification of the CBD-NMS system was written listing all system requirements, showing the graphical user interface and describing the behaviors of the system in normal operation and in case of exceptions. The main activity during the analysis phase was writing the Security Target First of all, we defined the Target of Evaluation (TOE). The TOE consists of multiple CBD-NMS applications communicating with each other. The Palm hardware is not part of the TOE. We made assumptions towards the security of the hardware.

The CBD-NMS system must ensure authentication during network object transaction. The security objectives are chosen to counteract the identified threats as in table 2. By requiring authentication (together with other requirements), we want to ensure that the threat has no impact on the CBD-NMS system. The TOE may not adequately protect cryptographic keys and data stored in the TOE, causing loss of confidentiality and integrity. After defining the security objectives, we identified the security functional requirements to substantiate the security objectives. In the CBD-NMS Security Target, the predefined CC security functional requirements (SFR) are used. The system uses cryptographic techniques to realize authentication. we began

writing the user guidance (class AGD). The first version of the manual showed the user interactions with the system. A software prototype of the CBD-NMS GUI was built and screen shots of the prototype were used in the user guidance. The CBD-NMS system does not provide system administration functionality. Therefore we rejected the CC requirement of writing an administrator guidance. When developing conformable to CC EAL2, a system test of the implementation with functional coverage is required (class ATE). Every test case defines pre-and post conditions of the test run, and the expected system interaction is documented using UML sequence grams as in figure 5.



**Fig. 5.** Sequence diagram for performance monitoring.

### 3.6 Design

During the design phase, we specified the architecture of the CBD-NMS system and the behavior of its components to provide a basis for the implementation. For this purpose, we created increasingly detailed design documents, starting from the information in the requirements specification and the Security Target. For EAL2, the CC requires two fairly abstract design documents, functional specification and high-level design. Thus, in the case study, these design documents were created to conform to the CC from the beginning on. We started with the functional specification which is a refinement of the TOE summary specification contained in the Security Target. It describes the security functions of the CBD-NMS system in a more detailed way. The more detailed specifications were not constrained by CC requirements at EAL2. The low-level design of the CBD-NMS system was specified using a UML class diagram

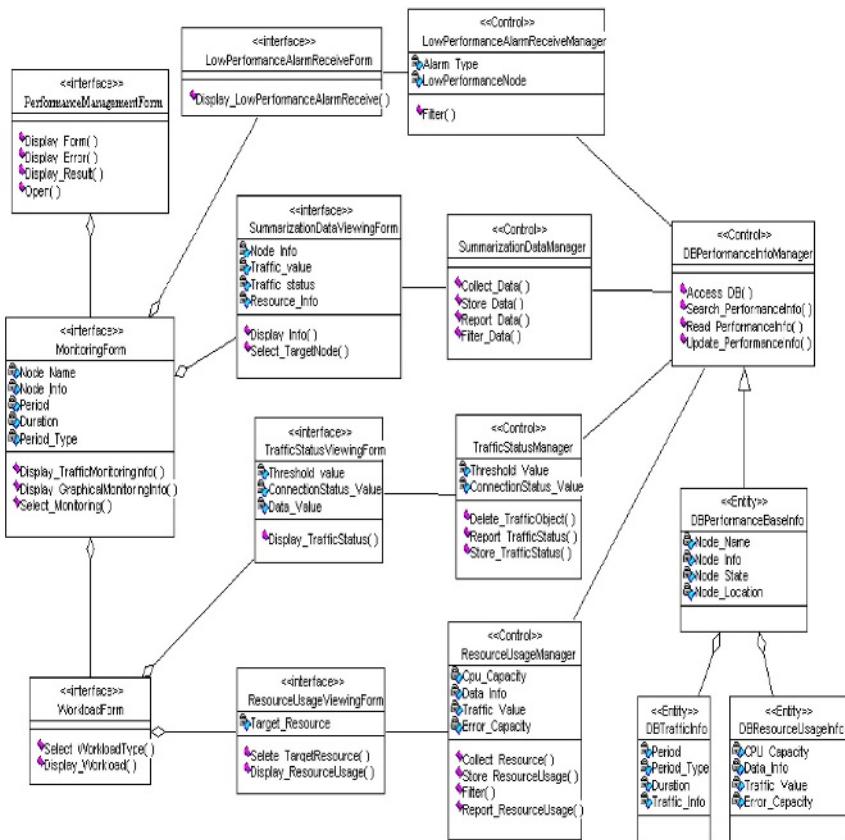


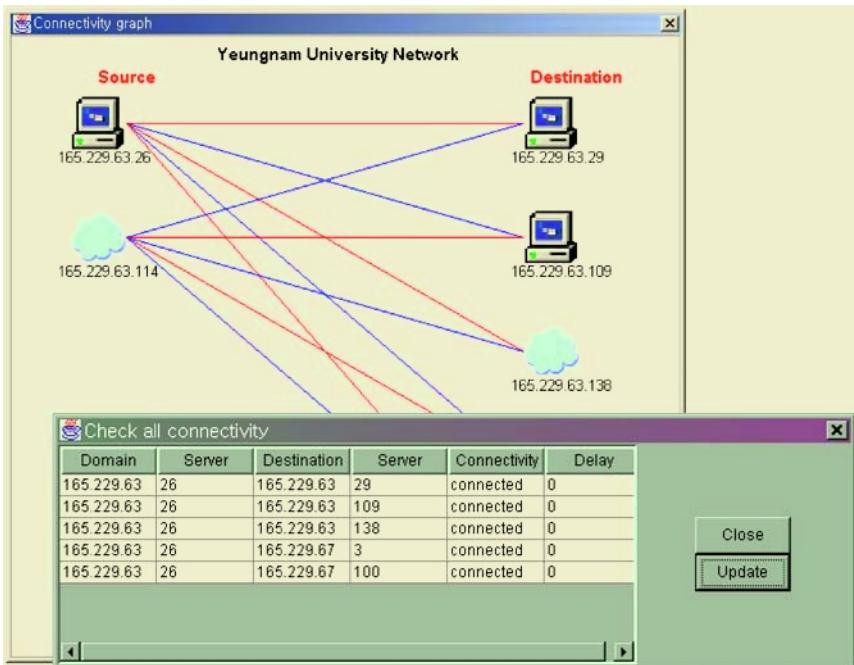
Fig. 6. Class diagram for CBD-NMS monitoring.

together with behavioral descriptions of the classes, giving a further refinement of the system structure specified in the high-level design as in figure 6.

When the implementation was finished, the system was tested using the test cases specified in the analysis phase. As the test cases had been specified at system level (user interface), the tests were carried out manually. The success or failure of each test case was documented in the test protocol, together with the TOE version, the responsible person, and date/time. The test case specifications and test coverage together with the test protocol represented the test documentation required by the CC. CBD-NMS provides assurance by an analysis of security functions, using a functional and interface specification, guidance documentation and the high level design for the TOE, to understand the security behavior. Independent testing of the TOE security functions supports the analysis, evidence of developer testing based on the functional specification, selective independent confirmation of the developer test results, strength of function analysis. With CBD-NMS implementation in this paper, Figure 7 shows connectivity check UI of connection management and perform check as assigning the source, destination and delay value.

## 4 Conclusions

In this paper, we have proposed an approach to the security characterization of software we show how integrate security aspects into the software engineering process. In addition, we also introduce our work on ensuring the reliability assurance in development process for Network Management System as TOE.



**Fig. 7.** Connectivity check of connection management UI.

The activities and documents from the Common Criteria are tightly intertwined with the system development, which improves the quality of the developed system and reduces the additional cost and effort due to high security requirements. component-based systems, and introduced our work on ensuring the integrity of software components as part of the infrastructural support for component based software engineering.

Our approach to security characterization is partially based on the international security evaluation standard, the Common Criteria, and aims to develop a composition model for software security. Our work on integrity assurance focuses on the development of a comprehensive authentication infrastructure for the development, distribution and use of software components.

## References

1. Common Criteria Project/ISO, "Common Criteria for Information Technology Security Evaluation Version 2.1 (ISO/IEC 15408)", <http://www.commoncriteriaportal.org/>, 1999.
2. "Information Technology-Software Life Cycle Process, (ISO/IEC 12207)", <http://standards.ieee.org/reading/ieee/std/>, 1997.
3. Ruben Prieto-Diaz, "The Common Criteria Evaluation Process," Commonwealth Information Security Center Technical Report, 2002.
4. Haeng Kon Kim, "Object modeling and pattern definition for the integrated network management based on CORBA", Proceeding of the Korea Multimedia Society, Vol.2, No.2, 1999.
5. Haeng Kon Kim, "A Study on the next generation Internet/Intranet Networking System Development", Technical Report, 2001.
6. Monika Vetterling, Guido Wimmel and Alexander Wisspeintner, "Requirements analysis: Secure systems development based on the common criteria: the PalME project", Proceedings of the tenth ACM SIGSOFT symposium on Foundations of software engineering, pp. 129 - 138, Nov. 2002.
7. "CC on Information Security System", KISA MIC, Korea 2002.
8. Haeng Kon Kim, "A Component Specification and Prototyping of Operator Interface System Construction for Network Management", SETC '2001, KIPS, Vol .5, No.1, 2001.
9. J.Han., "A comprehensive interface definition framework for software components", In Proceeding of the 1998 AsiaPacific Software Engineering Conference, Pages 110-117, Taipei, Taiwan, December 1998. IEEE Computer Society.
10. Jonathan Stephenson, "Web Services Architectures for Security," CBDI Journal, <http://www.cbdiforum.com/>, Feb. 2003.
11. Common Criteria Project/ISO, "Common Criteria for Information Technology Security Evaluation Version 2.1 (ISO/IEC 15408)," <http://www.commoncriteriaportal.org/cc/>, 1999.
12. "Information Technology-Software Life cycle Process, (ISO/IEC 12207)," <http://standards.ieee.org/reading/ieee/std/>, 1998.

## Author Index

- Bae, Doo-Hwan 148  
Bae, Ihn-Han 304  
Boehm, Barry 1  
Bures, Tomas 102  
  
Cain, Andrew 225  
Cha, Jung-Eun 266  
Chen, Tsong Yueh 225  
Cho, Il-Hyung 74  
Choi, Byoungju 168  
Choi, Sung-Hee 239  
  
Dosch, Walter 7  
  
Ferre, Xavier 349  
  
George, Laurent 51  
Grant, Doug 225  
  
Hong, Euyseok 209  
Hu, Gongzhu 332  
Hwang, Suk-Hyung 239  
Hwang, Sun-Myung 34  
  
Ishii, Naohiro 279  
Ito, Nobuhiro 279  
Iwata, Kazunori 279  
  
Jin, Jionghua 320  
Juristo, Natalia 349  
  
Kim, Chul 253  
Kim, Chul-Hong 266  
Kim, Eunhee 168  
Kim, Haeng-Kon 117, 364  
Kim, Jae-Sung 364  
Kim, Kapsoo 136  
Kim, Tai-Hoon 364  
Kim, Youngtak 253  
Kwon, Gihwon 290  
  
Lee, Byungjeong 87, 209  
Lee, Eunjoo 87  
Lee, Eunseok 320  
  
Lee, EunSer 185  
Lee, Jongwon 209  
Lee, Keun 185  
Lee, Kyung Whan 34, 185  
Lee, Roger Y. 117  
  
Magnussen, Sönke 7  
Martin, Steven 51  
McGregor, John D. 74  
Min, Sang-Yoon 148  
Minet, Pascale 51  
Miyazaki, Mayumi 279  
Moreno, Ana M. 349  
  
Oh, Jaewon 209  
Oh, Sun-Jin 304  
Olariu, Stephan 304  
  
Park, Dongchul 209  
Park, Geunduk 136  
Park, Ji-Hyub 34  
Paul, Raymond 4  
Plasil, Frantisek 102  
Poon, Pak-Lok 225  
  
Ramamoorthy, C.V. 2  
  
Şahinoglu, Mehmet 5  
Shin, Woochang 87, 136  
Song, Ki-Won 34  
Stafford, Judith A. 74  
  
Tang, Chunxia 332  
Tang, Sau-Fun 225  
Tse, T.H. 225  
  
Wu, Chisu 87, 136, 209  
  
Yang, Hae-Sool 117, 239  
Yang, Young-Jong 266  
Yeh, Raymond T. 3  
Yoon, Hoijin 168  
Yoon, Kyung-A 148