

José Cordeiro
AlpeshKumar Ranchordas
Boris Shishkov (Eds.)

Communications in Computer and Information Science

50

Software and Data Technologies

4th International Conference, ICSOFT 2009
Sofia, Bulgaria, July 2009
Revised Selected Papers

 Springer

Communications
in Computer and Information Science

50

José Cordeiro AlpeshKumar Ranchordas
Boris Shishkov (Eds.)

Software and Data Technologies

4th International Conference, ICSOFT 2009
Sofia, Bulgaria, July 26-29, 2009
Revised Selected Papers



Springer

Volume Editors

José Cordeiro
INSTICC and IPS
Setúbal, Portugal
E-mail: jcordeir@est.ips.pt

AlpeshKumar Ranchordas
INSTICC, Setúbal, Portugal
E-mail: alpesh@insticc.org

Boris Shishkov
IICREST, Sofia, Bulgaria
E-mail: b.b.shishkov@tudelft.nl

ISSN 1865-0929
ISBN 978-3-642-20115-8
DOI 10.1007/978-3-642-20116-5
Springer Heidelberg Dordrecht London New York

e-ISSN 1865-0937
e-ISBN 978-3-642-20116-5

Library of Congress Control Number: 2011925002

CR Subject Classification (1998): D.2, D.3, C.2.4, H.2, I.2.4

© Springer-Verlag Berlin Heidelberg 2011

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

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

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

Printed on acid-free paper

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

Preface

The present book includes extended and revised versions of a set of selected papers from the 4th International Conference on Software and Data Technologies (ICSOFT 2009), held in Sofia, Bulgaria, which was organized by the Institute for Systems and Technologies of Information, Communication and Control (INSTICC), co-organized by the Interdisciplinary Institute for Collaboration and Research on Enterprise Systems and Technology (IICREST), in cooperation with the Bulgarian Academy of Sciences and the Technical University of Sofia and technically co-sponsored by the Workflow Management Coalition (WfMC).

The purpose of ICSOFT 2009 was to bring together researchers and practitioners interested in information technology and software development. The conference tracks were “Enterprise Software Technology,” “Software Engineering,” “Distributed Systems,” “Data Management” and “Knowledge-Based Systems.”

Being crucial for the development of information systems, software and data technologies encompass a large number of research topics and applications: from implementation-related issues to more abstract theoretical aspects of software engineering; from databases and data-warehouses to management information systems and knowledge-based systems; next to that, distributed systems, pervasive computing, data quality and other related topics were included in the scope of this conference.

ICSOFT 2009 received 212 paper submissions from 50 countries in all continents. To evaluate each submission, a double-blind paper evaluation method was used: each paper was reviewed by at least two internationally known experts from the ICSOFT Program Committee. Only 27 papers were selected to be published and presented as full papers, i.e., completed work (8 pages in the proceedings / 30 min oral presentations), 53 additional papers describing work-in-progress were accepted as short papers for 20 min oral presentation, leading to a total of 80 oral paper presentations. Another 33 papers were selected for poster presentation. The full-paper acceptance ratio was thus 13%, and the total oral paper acceptance ratio was 38%.

We hope that you will find these papers interesting, considering them a helpful reference in the future when addressing any of the research areas mentioned above.

July 2010

José Cordeiro
Alpesh Ranchordas
Boris Shishkov

Conference Committee

Conference Co-chairs

José Cordeiro

Polytechnic Institute of Setúbal /
INSTICC, Portugal

AlpeshKumar Ranchordas

INSTICC, Portugal

Program Chair

Boris Shishkov

IICREST / Delft University of
Technology, The Netherlands

Organizing Committee

Patrícia Alves

INSTICC, Portugal

Sérgio Brissos

INSTICC, Portugal

Helder Coelhas

INSTICC, Portugal

Vera Coelho

INSTICC, Portugal

Andreia Costa

INSTICC, Portugal

Bruno Encarnação

INSTICC, Portugal

Bárbara Lima

INSTICC, Portugal

Raquel Martins

INSTICC, Portugal

Elton Mendes

INSTICC, Portugal

Carla Mota

INSTICC, Portugal

Vitor Pedrosa

INSTICC, Portugal

Mónica Saramago

INSTICC, Portugal

José Varela

INSTICC, Portugal

Pedro Varela

INSTICC, Portugal

Program Committee

Jemal Abawajy, Australia

Toshiaki Aoki, Japan

Silvia Abrahão, Spain

Kejiro Araki, Japan

Alain Abran, Canada

Gabriela Noemí Aranda, Argentina

Muhammad Abulaish, India

Farhad Arbab, The Netherlands

Hamideh Afsarmanesh,

Cyrille Artho, Japan

The Netherlands

Colin Atkinson, Germany

Jacky Akoka, France

Xiaoying Bai, China

Markus Aleksy, Germany

Mortaza S. Bargh, The Netherlands

Daniel Amyot, Canada

Joseph Barjis, The Netherlands

Kenneth Anderson, USA

Zeki Bayram, Cyprus

- Fevzi Belli, Germany
Alexandre Bergel, Chile
Árpád Beszédes, Hungary
Wladimir Bodrow, Germany
Marello Bonsangue, The Netherlands
Lydie du Bousquet, France
Mark Van Den Brand,
The Netherlands
Lisa Brownsword, USA
Fergal Mc Caffery, Ireland
Gerardo Canfora, Italy
Mauro Caporuscio, Italy
Cinzia Cappiello, Italy
Licia Capra, UK
Sergio de Cesare, UK
Krzysztof Cetnarowicz, Poland
Jinjun Chen, Australia
Kung Chen, Taiwan
Shiping Chen, Australia
Chia-Chu Chiang, USA
Peter Clarke, USA
Rem Collier, Ireland
Kendra Cooper, USA
Alexandra Cristea, UK
Sergiu Dascalu, USA
Steven Demurjian, USA
Giovanni Denaro, Italy
Oscar Dieste, Spain
María J. Domínguez-Alda, Spain
Jing Dong, USA
Juan C. Dueñas, Spain
Philippe Dugerdil, Switzerland
Jürgen Ebert, Germany
Raimund K. Ege, USA
Fikret Ercal, USA
Onyeka Ezenwoye, USA
Cléver Ricardo Guareis de Farias,
Brazil
Massimo Felici, UK
Rudolf Ferenc, Hungary
Juan Fernandez-Ramil, UK
Gianluigi Ferrari, Italy
Kehan Gao, USA
Jose M. Garrido, USA
Dragan Gasevic, Canada
Mouzhi Ge, Ireland
Nikolaos Georgantas, France
Paola Giannini, Italy
J. Paul Gibson, France
Itana Gimenes, Brazil
Swapna Gokhale, USA
Juan Carlos Granja, Spain
Christophe Gravier, France
Des Greer, UK
Klaus Grimm, Germany
Giancarlo Guizzardi, Brazil
Slimane Hammoudi, France
Christian Heinlein, Germany
Markus Helfert, Ireland
Jose Luis Arciniegas Herrera,
Colombia
Jang-eui Hong, Korea, Republic of
Nien-Lin Hsueh, Taiwan
Ilian Ilkov, The Netherlands
Ivan Ivanov, USA
Yong-Kee Jun, Korea, Republic of
Wan Kadir, Malaysia
Sanpawat Kantabutra, Thailand
Roger (Buzz) King, USA
Mieczyslaw Kokar, USA
Jun Kong, USA
Rainer Koschke, Germany
Walter Kusters, The Netherlands
Martin Kropp, Switzerland
Tei-wei Kuo, Taiwan
Patricia Lago, The Netherlands
Konstantin Laufer, USA
Yu Lei, USA
Raimondas Lencevicius, USA
Hareton Leung, China
Mary Levis, Ireland
Kuan-Ching Li, Taiwan
Hua Liu, USA
K.S. Low, Singapore
Andrea de Lucia, Italy
Christof Lutteroth, New Zealand
Ricardo J. Machado, Portugal
Broy Manfred, Germany
Yannis Manolopoulos, Greece
Eda Marchetti, Italy

Ami Marowka, Israel
Katsuhisa Maruyama, Japan
Tommaso Mazza, Italy
Bruce McMillin, USA
Karl Meinke, Sweden
Atif Memon, USA
Jose Ramon Gonzalez de Mendivil,
Spain
Raffaela Mirandola, Italy
Dimitris Mitrakos, Greece
Mattia Monga, Italy
Sandro Morasca, Italy
Henry Muccini, Italy
Paolo Nesi, Italy
Jianwei Niu, USA
Rory Oconnor, Ireland
Pasi Ojala, Finland
Flavio Oquendo, France
Vincenzo Pallotta, Switzerland
Witold Pedrycz, Canada
Patrizio Pelliccione, Italy
Massimiliano Di Penta, Italy
Thoa Pham, Ireland
Martin Pinzger, The Netherlands
Pascal Poizat, France
Lori Pollock, USA
Andreas Polze, Germany
Peter Popov, UK
Christoph von Praun, Germany
Rosario Pugliese, Italy
Rafa Al Qutaish, Jordan
Jolita Ralyte, Switzerland
T. Ramayah, Malaysia
Anders Ravn, Denmark
Marek Reformat, Canada
Arend Rensink, The Netherlands
Werner Retschitzegger, Austria
Claudio de la Riva, Spain
Colette Rolland, France
Gustavo Rossi, Argentina
Gunter Saake, Germany
Krzysztof Sacha, Poland
Francesca Saglietti, Germany
Isabel Seruca, Portugal
Marian Fernández de Sevilla, Spain
Beijun Shen, China
Yanfeng Shu, Australia
Marten Van Sinderen, The Netherlands
Harvey Siy, USA
Yeong-tae Song, USA
George Spanoudakis, UK
Peter Stanchev, USA
Nenad Stankovic, USA
George K. Thiruvathukal, USA
Laurence Tratt, UK
Sergiy Vilkomir, USA
Christiane Gresse Von Wangenheim,
Brazil
Hironori Washizaki, Japan
Jens H. Weber-Jahnke, Canada
Edgar Weippl, Austria
Ing Widya, The Netherlands
Qing Xie, USA
Bin Xu, China
Haiping Xu, USA
Hongji Yang, UK
Stephen Yang, Taiwan
Tuba Yavuz-kahveci, USA
I-Ling Yen, USA
Fatiha Zaidi, France
Gregor Zellner, Germany
Xiaokun Zhang, Canada
Zhenyu Zhang, Hong Kong
Hong Zhu, UK
Elena Zucca, Italy

Auxiliary Reviewers

Jeroen Arnoldus, The Netherlands
Matt Bone, USA
Stefano Busanelli, Italy
Luigi Cerulo, Italy
Enrique Fernandez, Argentina
Ralf Gitzel, Germany
Anna Grimán, Venezuela
Maurice Hendrix, UK
Joseph Kaylor, USA
Jérémy Lardon, France
Giuseppe Antonio Di Lucca, Italy
Ivano Malavolta, Italy
Paolo Medagliani, Italy

Fei Niu, Sweden
Joseph Okika, Denmark
George Pallis, Cyprus
Ignazio Passero, Italy
Plamen Petrov, USA
Eleftherios Tiakas, Greece
Tom Verhoeff, The Netherlands
Saleem Vighio, Denmark
Tien-Hsiung Weng, Taiwan
Matthew Wojtowicz, USA
Maria Zimakova, The Netherlands
Eugenio Zimeo, Italy

Invited Speakers

Roel Wieringa
Jorge Cardoso
Kecheng Liu
Mihail Mihaylov Konstantinov

University of Twente, The Netherlands
SAP AG, Germany
University of Reading, UK
University of Architecture, Civil
Engineering and Geodesy, Bulgaria

Table of Contents

Invited Papers

IoS-Based Services, Platform Services, SLA and Models for the Internet of Services	3
<i>Jorge Cardoso, Matthias Winkler, Konrad Voigt, and Henrike Berthold</i>	
Pragmatic Web Services: A Semiotic Viewpoint	18
<i>Kecheng Liu and Adrian Benfell</i>	

Part I: Enterprise Software Technology

Unified Hosting Techniques for the Internet of Tradeable and Executable Services	35
<i>Josef Spillner, Iris Braun, and Alexander Schill</i>	
Service Differentiation in Multi-tier Application Architectures	46
<i>Mursalın Habib, Yannis Viniotis, Bob Callaway, and Adolfo Rodriguez</i>	
Lessons Learned on the Development of an Enterprise Service Management System Using Model-Driven Engineering	59
<i>Rodrigo García-Carmona, Juan C. Dueñas, Félix Cuadrado, and José Luis Ruiz</i>	

Part II: Software Engineering

Checking Regulatory Compliance of Business Processes and Information Systems	71
<i>Motoshi Saeki, Haruhiko Kaiya, and Satoshi Hattori</i>	
A Decision Support Scheme for Software Process Improvement Prioritization	85
<i>Arne Beckhaus, Lars M. Karg, Christian A. Graf, Michael Grottke, and Dirk Neumann</i>	
Development of a Family of Personalized Mobile Communicators	94
<i>Miguel A. Laguna and Bruno González-Baixauli</i>	
Reverse Generics: Parametrization after the Fact	107
<i>Alexandre Bergel and Lorenzo Bettini</i>	

A Calculus of Agents and Artifacts	124
<i>Ferruccio Damiani, Paola Giannini, Alessandro Ricci, and Mirko Viroli</i>	
Using Trace to Situate Errors in Model Transformations	137
<i>Vincent Aranega, Jean-Marie Mottu, Anne Etien, and Jean-Luc Dekeyser</i>	
Design of SOA Services: Experiences from Industry	150
<i>Susanne Patig</i>	

Part III: Distributed Systems

Division of Water Supply Systems into District Metered Areas Using a Multi-agent Based Approach	167
<i>Joaquín Izquierdo, Manuel Herrera, Idel Montalvo, and Rafael Pérez-García</i>	
Rateless Codes for Reliable Data Transmission over HomePlug AV Based In-Home Networks	181
<i>J.P. Muñoz-Gea, P.J. Piñero-Escuer, J. Malgosa-Sanahuja, P. Manzanares-Lopez, and J.C. Sanchez-Aarnoutse</i>	
Replication Strategies for Business Objects in SOA	192
<i>Michael Ameling, Bernhard Wolf, Thomas Springer, and Alexander Schill</i>	
A Hybrid Approach for Database Replication: Finding the Optimal Configuration between Update Everywhere and Primary Copy Paradigms	205
<i>M. Liroz-Gistau, J.R. Juárez-Rodríguez, J.E. Armendáriz-Íñigo, J.R. González de Mendúvil, and F.D. Muñoz-Escóí</i>	
Educational Resource Scheduling Based on Socio-inspired Agents	218
<i>Juan I. Cano, Eloy Anguiano, Estrella Pulido, and David Camacho</i>	

Part IV: Data Management

Managing Risks by Integrity Constraints and Integrity Checking	233
<i>Hendrik Decker</i>	
Energy Efficient Data Sorting Using Standard Sorting Algorithms	247
<i>Christian Bunse, Hagen Höpfner, Suman Roychoudhury, and Essam Mansour</i>	

Part V: Knowledge-Based Systems

Analysis of Emergent and Evolving Information: The Agile Planning Case	263
<i>Rasmus Rosenqvist Petersen and Uffe Kock Wiil</i>	
Emotion Based User Interaction in Multimedia Educational Applications.....	277
<i>Efthymios Alepis, Maria Virvou, and Katerina Kabassi</i>	
Author Index	291

Invited Papers

IoS-Based Services, Platform Services, SLA and Models for the Internet of Services

Jorge Cardoso¹, Matthias Winkler², Konrad Voigt², and Henrike Berthold²

¹ CISUC/University of Coimbra, Portugal

² SAP Research CEC, Chemnitzer Strasse 48, 01187 Dresden, Germany

Abstract. The Internet of Services (IoS) embraces new business and technological models that can radically change the way day-to-day services are provisioned and contracted. Within the IoS, services which were traditionally not created based on an established science and operated without a systematic approach, will be digitalized using proper methodologies, standards, and tools. The digital specification of services will enable their trading over the Internet using electronic marketplaces. This paper provides four main contributions to the IoS research: a definition and characterization for the concept of IoS-based service, the role and importance of platform services for the IoS, the challenges of managing SLA dependencies between IoS-based services in compositions, and a model-based approach for service engineering that can be used to design IoS-based services.

1 Introduction

Nowadays, all industrialized countries have become service-based economies in terms of the distribution of the people employed in the service sector [1]. While the first services were certainly delivered by humans to humans, the advances in computer systems over the past sixty years allowed computers to deliver services to humans. Information technologies (IT) have significantly contributed to the evolution of services. Over the years, each generation of innovation has created solutions that enable to automatically execute activities that were once done by human beings [2].

For example, Automated Teller Machines (ATM), introduced in the 70s, enabled banks to reduce costs by decreasing the need for tellers. Even with the expensive cost of IT in the late 70s and early 80s, the cost of automated processing with ATM was less than the expenditure of hiring and training a teller to carry out the same activity. In this case, a complex and specialized machine was developed to perform the same activity once executed by a human. As technology brought the automation to sophisticated machines, the number of workers required to execute many activities was gradually reduced. Nowadays, a broad spectrum of services is being replaced by automated machines. For example, undertaking a trip by train has traditionally required passengers to purchase a ticket from an office and show it for inspection when required by the train operator. As a response to technological development, automated dispensers have reduced the need to queue in order to purchase a ticket before a trip and, therefore, enable faster journeys.

The historical perspective and evolution of services has not only been confined to the use of machines to automate services. The emergence of the Internet, allied with the

World Wide Web, has allowed a remote and generalized interaction between humans and computers. The technological developments in the late 90s have pushed the notion of *service* to *Web service* with the objective of supporting interoperable computer-to-computer interactions over a data network. This type of interaction required services to be autonomous and platform-independent, and needed services to be described, published, discovered and orchestrated using standard protocols for the purpose of building distributed solutions. The emphasis was on the definition of interfaces from a technical and programming perspective. The objective was on distributed systems communication, since Web services provide a technological solution to enable enterprise transaction systems, resource planning systems and customer management systems to be accessed programmatically through a digital network.

The IoS takes these developments one step further. So far, the use of services (i.e. Web services) has been restricted to IT professionals and to IT departments inside organizations. The Internet of Service targets to investigate and develop new theories, models, architectures and technologies to provide efficient and effective solutions that enable also non-professional users to create, trade and consume services. Furthermore, the notion of service is not limited to IT-based or technical services, but also to real world or day-to-day services. This larger spectrum of services immediately foresees a methodical study on how these services can be represented and modeled digitally.

This paper explores four important topics for the Internet of Services (these results are part of the output of the TEXO project which is part of the THESEUS program¹) The first topic addresses the evolution of the concept of service, the economical value and the major characteristics of IoS-based services. These aspects are important in order to fully understand why the IoS and its services are fundamentally distinct from previous endeavors in the area of service provisioning. The second topic depicts a taxonomy for the various platform services that will be made available to providers and that will be used to design and operate IoS-based services. As the name indicates, platform services are provided by the underlying platform where IoS-based services are provisioned. The third topic tackles the support for managing SLA dependencies with a focus on IoS-based services that have a process composition in their backend. The violation of a particular SLA contract in a composition can affect related SLA and, therefore, dependencies and their impact need to be studied. Finally, the last topic covers the design of IoS-based services using a model-based approach. The design method proposed is part of the lifecycle of Service Engineering and relies on the integration of models that describe the various facets of a IoS-based service using efficient matching algorithms.

2 The Concept, Value and Characteristics of Services

Research in service science can often lead to confusion due to the simple fact that different communities use the same term to refer to conceptually distinct services. To avoid any ambiguity, this section will clarify key concepts associated with the term service upfront. The value that IoS-based services can bring to worldwide economies and the intrinsic characteristics of services when compared to products will also be reviewed.

¹ <http://theseus-programm.de/en-US/home/default.aspx>

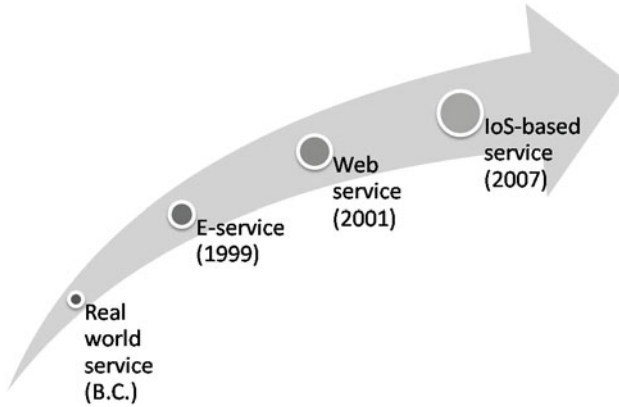


Fig. 1. Terms most frequently associated with the concept of service

2.1 The Concept of Service

Baida et al. [3] have identified that the terms *service*, *e-service* and *Web service* actually address related concepts from different domains such as computer science, information science and business science.

In computer science, the terms *service*, *e-service* and *web service* are generally used to identify an autonomous software component that is uniquely identified by a universal resource identifier (URI) and that can be accessed using standard Internet protocols, such as the simple object access protocol (SOAP) and the hypertext transfer protocol (HTTP), and languages such as the extensible markup language (XML). Hull et al. [4] have used the term *e-service* to describe the functionalities and characteristics of a *Web service*. They have defined an *e-service* as *a collection of network-resident software services accessible via standardized protocols, whose functionality can be automatically discovered and integrated into applications or composed to form more complex services*. In information science, services are a means of delivering value to customers by facilitating outcomes the customers want to achieve without the ownership of specific costs and risks [5]. Outcomes are created from the execution of tasks or activities under a set of constraints. In business science, a service is a set of intangible activities that generally take place in interactions between a provider and a consumer. The emphasis is not on standards, protocols or software, but on the study of how consumer experience with services can be evaluated and improved.

A deeper understanding of the terms associated to the concept of service is necessary in order to conceptually craft a common frame of reference. Such a shared understanding will help stakeholders involved in building enterprise wide solutions based on services for the IoS. Therefore, there is the need to identify the terms most often associated with the concept of service that have been introduced over time by the research community and by the industry. The four most relevant terms, illustrated in Figure 1, are real world services (day-to-day services), *e-services*, *Web services* and, more recently, *IoS-based services*. We introduce the term *IoS-based service* to differentiate a new type of service that will be represented with digital structures, specifications and standards to be traded on the Internet of Services.

Real World Service. The term real world service (i.e., a day-to-day service or simply a service) is used to refer to any type of service that can be found in society. Because of their diversity and heterogeneity, real world services, have traditionally been difficult to define. Kotler [6] defines a service as *any activity or benefit that one party can give to another that is essentially intangible and does not result in the ownership of anything*. Its production may or may not be tied to a physical product. For Payne [7], a service is an activity that has an element of intangibility associated with it and which involves the service provider's interaction either with the customers or with the property belonging to the customer.

E-services. E-services are services for which data networks, such as the Internet, are used as a channel that allow consumers to interact with remote services. Virtually any service can be transformed into an e-service if it can be invoked via a data network. E-services are independent of the specification language used to define its functionality, non-functional properties or interface. As with real world services, the definition of e-service is fairly broad. The main requirement is that the service must allow a remote invocation and interaction using a data network as a communication channel. The specification of a service from a business and technical perspective is not mandatory.

Web Services. Web services allow software applications to easily communicate, independently of the underlying computing platform and language. The use of Web services is substantially less complex than the use of prior solutions for creating interoperable distributed systems. Heterogeneous, autonomous and distributed applications have been a vital field since computing shifted from jobs running on centralized mainframe computers to networked computers. Previous technologies that covered the same objectives as Web services included Remote Procedure Call (RPC), Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM) and Java Remote Method Invocation (JRMII). These technologies had drawbacks that were considered significant when developing distributed application, such as incompatibility across vendors' implementations, and complexity and cost of solutions.

IoS-based Service. The term "IoS-based service" is used to identify services provided through the Internet. Two main characteristics make IoS-based services distinct from previous services. On the one hand, this notion of service is not limited to IT-based services, but also to real world or day-to-day services. On the other hand, the stakeholders of such services, from the provisioning and consumption side, are not only IT professionals but also non-professional users. As a result, IoS-based services serve a dual purpose since they can be utilized directly by consumers, but they can also be invoked by technical systems to access business functionality which is provided remotely by business providers. An IoS-based service model defines a view on services that is provision-oriented and service-centric. An important feature of a digital service model is a separation of characteristics in terms of business, operational and technical perspectives.

Compared to previous approaches to support services – which were mainly implemented as pieces of software (e.g. Web services) – developing solutions for the IoS is more elaborate since real world services have very specific characteristics. While e-services and Web services are seen mainly as technological entities, the Internet of

Services will take the representation of services one step further. IoS-based services will combine and correlate business, operational and IT aspects into service descriptions.

2.2 The Economical Value of Services

The intense competition of economies and the globalization of worldwide markets in conjunction with the generalization and expansion of IS and IT have opened up significant opportunities for the conception of new specialized services. Services are becoming quickly more productized. Providers are focusing on services for increased differentiation and creation of consumer value as a source of competitive advantage.

Recently, the concept of service has acquired a renewed importance since after several years of public debate, the European Parliament has approved the service directive [8]. This directive intends to enhance competition by removing restrictions on cross-border market access for services in Europe. The implications of this measure for businesses and the IT community are enormous since the service sector represents more than 70% of the Gross National Product and the directive can amplify the consumption of services in the European Union by 0.6% (37 billion Euros) [9]. Figure 2 illustrates the gross value added of services in Germany in 2005 provided by the Statistisches Bundesamt². In Germany, the service sector represents 69.4% of the gross value added.

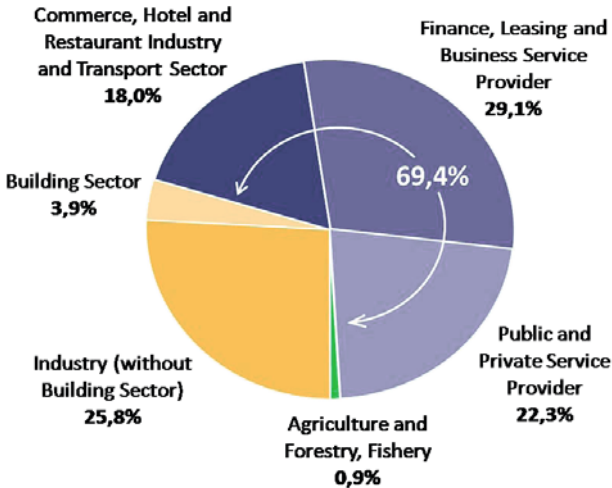


Fig. 2. Gross value added of services in Germany

Services seem to be the new hub for most economies. Infrastructure services such as transportation and communication are fundamental building blocks which link to all other sectors. In most countries, one of the largest and most important providers of services is the government which operates in sectors such as water management, public safety and basic healthcare system.

Based on the economical value and importance of services, one question that immediately arises is how can the Internet provide a solution to create and enable a genuine

² <http://www.destatis.de/>

market for the trade of cross-border services? Since the Internet is now an integral ingredient of the fabric of worldwide societies, economies and commerce, it can intuitively provide a fundamental infrastructure to enable the realization of the IoS.

2.3 Intrinsic Characteristics Services

Before proposing or building a solution for the IoS and for IoS-based services, it is fundamental to understand the nature of real world services since it is this type of services that will be digitalized and represented with proper models to enable their trading over the Internet. Real world services are often known to have one or more of the following characteristics: intangible, inseparable, immersive, bipolar, variable, ostensible, long-running, decoupled, perishable and qualitative.

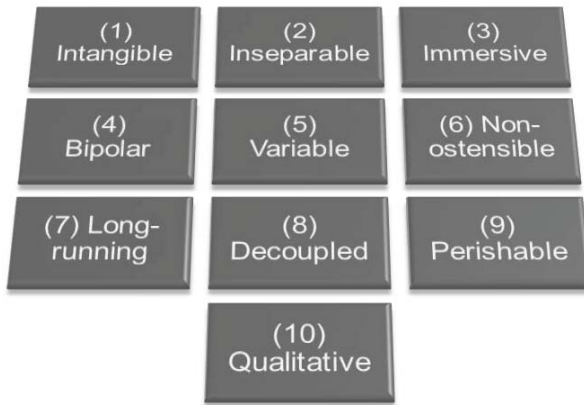


Fig. 3. Characteristics of services

Intangible (1). Services are intangible since they do not have a material existence. One can physically touch or view a product but most services are intangible. Nonetheless, it is often possible to see and evaluate the results that arise from the execution of a service. For example, it is not feasible to touch a legal advice or a doctor consultation. Therefore, it is difficult to create suitable specifications to model and to define attributes to objectively describe services. As such, research needs to be undertaken to determine which fundamental aspects and characteristics of real world services should be present in IoS-based service models and descriptions.

Inseparable (2). The provisioning and consumption of services occurs frequently in parallel. This implies that a rigorous match between supply and demand must be achieved. Otherwise, services are lost or consumers are queued and need to wait for service availability. This characteristic leads to a challenging research question: How can marketplaces of the IoS provide mechanisms to match between supply and demand efficiently?

Immersive (3). Services are often executed in collaboration, involving providers and consumers. This implies that in many cases it is difficult to determine the parties responsible for the degree of success or failure of a service. Therefore, when IoS-based

services are executed, it is important to define service level agreements (SLA) that account for both parties. But in case of service provisioning failure, since many services are executed in collaboration, how can responsibilities be determined?

Bipolar or Hybrid (4). Services are often executed by a blend of human and technological resources. While approaches to monitor purely technological resources are already available, solutions to monitor human involvement in services' execution and the complex relationship between the human and technological dimensions have not been studied in the context of highly distributed, autonomous and heterogeneous settings, such as the IoS. As a result, the following research question arises: How to create mechanisms that account for the monitoring of technological resources with the individual monitoring of human resources?

Variable (5). Products have a high degree of standardization, while services are very often tailor-made and are, therefore, heterogeneous. Organizations differentiate themselves in offering products and services, but the variations between similar products of different providers are less prominent than the variations between services. The quality and consistency of services is subject to a considerable variability since they are often delivered by humans. Human attitudes (such as behavior, cognitive skills, emotions, style, etc.) have a high variability since they are difficult to control, manage and evaluate. For the IoS, this characteristic brings the need to devise new variability models for services, perhaps based on the variability models which have already been developed for products.

Ostensible Ownership (6). The ownership between products and services is distinct. A stock can be called a financial product that the provider owns. A stock order may be placed by a consumer which might result in a transaction later on. When the transaction is completed, the ownership is transferred to the consumer. On the other hand, it is not possible to own a service. Its possession is termed as an ostensible ownership. As a result, the IoS needs to enable providers to have the ostensible ownership of IoS-based services and enable them to remain in their control and management.

Long-running (7). Generally, the trading of products requires a low level of interaction between providers and consumers. For example, when a consumer buys a book at Amazon.com there is only one interaction point in time: the action of purchasing the book. In exceptional scenarios, the consumer may contact the provider in case of non-delivery of the product. On the other hand, services are often executed by a back-end business process which involves human interaction over time until the service is completed. For example, a service contracted to translate a book from German to English may run for several weeks and require a significant interaction between the translator and the writer. This brings humans, relationships between people, processes and activities to be an integral part of services. Therefore, IoS-based services need to account for the definition and representation of long-running business processes which include personal interactions (e.g., face-to-face or a telephone interaction) between providers and consumers.

Decoupled (8). A simplified lifecycle of a service includes generally five main phases: discovery, selection, invocation, execution and termination. In order to capture the full potential of services, consumers must have access to dynamic discovery mechanisms.

Once a set of services is discovered, a selection is made and the selected service is invoked. Finally, the service is executed and terminates. These five phases can be carried out only with human involvement (humans add value in the form of labor, advice and skills), with a conjunction of humans and IT, or resorting purely on automated processing. Therefore, in the IoS, each phase is decoupled and may position itself anywhere in the spectrum of services executed solely by humans, on the one side, or purely automated on the other side. Here again, the representation of human and IT involvement needs to be equated when modeling IoS-based services.

Perishable (9). Since services have a propensity to be intangible, it is usually not possible to store them. As a consequence, services are perishable and unused capacity cannot be stored for future trade. For example, if a car is not sold today it can be sold tomorrow but spare seats on an airplane cannot be transferred to the next flight. Not being able to store services brings a challenge for electronic marketplaces since new management methods for service capacity are required.

Qualitative (10). In manufacturing industries, measures determining the quality of products are generally quantitative. On the other hand, the quality of a service is generally qualitative. The physical evidence or the tangible products that originate from service execution can provide indications that allow measuring a service quality. This characteristic is again a challenge for the IoS. How to identify which aspects of a service execution can be used to evaluate the quality of services quantitatively and qualitatively? Furthermore, the perceived service quality often results from consumers comparing the outcome of a service against what they expected to receive. Thus, how can consumers' expectations be properly captured and managed in the IoS?

All these intrinsic characteristics of services need to be explored and equated when real world services are digitalized into electronic models to be advertized in marketplaces and purchased remotely by consumers using the Internet.

3 Platform Services

Platform services provide functionalities as a common infrastructure that can be used by IoS-based services in business value networks. We divide platform services into four groups: core business services, design support services, execution support services, and consumer support services. The elements of this classification are not orthogonal and one specific platform service can be classified under one or more types.

Core business services provide base functionalities required to complete a value-generating service to an IoS-based service. A value-generating service is the concrete service that generates a value for its consumer. An IoS-based service is a composite service that contains the value-generating service and core business services that are required to trade them. The set of core business services should contain services for payment, billing, security, and community rating. However, depending on the concrete service delivery platform, the set of these services can vary. The more core business services are available, the more business models can be supported. A core business service often provides a customized user interface that is integrated in the cockpit of service

consumers. As a result, a consumer has to provide the required data for a core service to perform. For example, in a payment core service, the consumer must specify the kind of payment for the use of an IoS-based service.

Execution support services extend the functionality of the runtime environment. Their usage has to be specified at design time. Examples are adaptation services, monitoring services, and efficient data propagation services. An adaptation service automatically adapts an IoS-based service to changes in the runtime environment or consumer requirements without affecting the functional results. A monitoring service measures a number of runtime characteristics and can notify service consumers, service providers or platform providers about anomalies that occur at the technical and business level. An efficient data propagation service allows services that process large data volumes to exchange these data in an efficient way.

Consumer support services provide a functionality to retrieve services that fulfill the consumer's needs and requirements. Examples are services to search and select appropriate IoS-based services. Consumer support services can be accessed via the consumer cockpit . They use service description repositories for the search functionalities.

Design support services help the design of IoS-based services. They typically have a design component that supports business process modeling and a runtime component that runs as part of the overall process. Examples of design support services include message matching services and data integration services. The message matching service serves as an example of a platform service which is used to map complex message schemas between IoS-based services that form compositions. Therewith, it enables the composition of independently developed services offered on a service marketplace. The message matching service has two main components: the mapping component and the translation component. The mapping component is the design component. It automatically calculates a proposal for a mapping between two message schemas. A graphical user interface is provided for that component which presents a proposed mapping and supports its manual completion to a final mapping. The translation component is the runtime component. It uses the specified mapping to translate messages.

4 Managing Dependencies in IoS-Based Service Compositions

One goal of the IoS vision is the creation of service compositions out of atomic IoS-based services provided by different service providers. The different atomic IoS-based services are composed in a way that they form more complex functionality, i.e. they are implicitly collaborating. These compositions are then offered to consumers via a marketplace. The provisioning of atomic as well as composite IoS-based services is regulated by service level agreements (SLA) which are negotiated between the service providers and consumers of the respective services.

An important challenge of this scenario is the management of such service compositions to ensure that the atomic services are working together in a proper way to achieve the overall goal of the composition. This management task is handled by the composite service provider who selects the atomic services and negotiates the SLAs with atomic service providers and composite service consumers. Managing service compositions

is a challenging task due to the fact that the collaborating services have dependencies on each other. This leads to failure propagation. Also, the different SLAs need to be negotiated in a way which ensures proper collaboration between services. Changes to an SLA after the initial negotiation may require changes to other SLAs. In the following sections we outline the problem space of dependencies in service compositions and present our approach to managing these dependencies.

4.1 Introducing Dependencies

A service dependency is a directed relation between services. It is expressed as a 1-to-n relationship where one service (dependant) depends on one or multiple services (antecedent). A service S_1 is dependent on a service S_2 if the provisioning of service S_1 is conditional to the provisioning of service S_2 , e.g. if a property of service S_1 is affected by a property of S_2 .

A service can be dependent on another service with regard to different aspects. A dependency occurs e.g. when one service provides something (e.g. data or goods) which is needed by another service to provide its functionality. A second example would be that the quality of service (QoS) of a composite service depends on the QoS of its atomic services. Dependencies occur either between atomic services in a process (horizontal dependencies) or between the composite service and atomic services (vertical dependencies) [10]. Horizontal dependencies mainly occur due to provider-consumer relationships or simultaneity constraints between services while vertical dependencies occur due to task-subtask relationships between the composite service and the atomic services [11]. Distinguishing the underlying causes of dependencies is important as they form the base for the dependency discovery strategy (see section 4.3).

4.2 Approach to Managing Dependencies

In order to manage service dependencies we developed an approach for capturing dependency information at design time in a dependency model and evaluating this model at runtime when problems occur or an SLA needs to be renegotiated. The approach follows a lifecycle of four steps: creation, validation, usage, retirement. During the first step (*creation*) the dependency information captured in a dependency model. Information about service dependencies is not available explicitly but rather implicitly in the process description and the related SLAs. To make it available explicitly for easier handling at runtime, the process description and SLA information are analyzed and dependency information is extracted and captured in a dependency model. The dependency model is then stored in a dependency repository. During the second step (*validation*) the dependency model is validated to check if the SLAs have been negotiated in a way that the services can successfully collaborate to achieve the composite service goals. During the third step (*usage*) the dependency model is evaluated at runtime in the context of occurring events such as SLA violations during monitoring or requests for SLA renegotiation. The goal of this evaluation is to determine effects of the current event on other services (i.e. if the SLA for service S_1 is changed, which other services will be affected). Finally, during the fourth step (*retirement*) the dependency model is terminated once the composite service is terminated and the composite service SLA is expired. During this step the dependency model is removed from the dependency repository.

At the base of this approach there is a dependency model. A meta-model for capturing dependencies was developed for this purpose and a model editor for creating dependency model instances was implemented based on this meta-model [12]. While the model editor allows the full specification of dependency model instances, we also developed a model creation approach which partially automates this procedure.

4.3 Dependency Model Creation

The process of creating a dependency model is separated into two major steps. An initial model is generated automatically by an algorithm which analyses the process description and SLAs. In the second step the model can be refined by manual changes using a model editor. While the first step enables a more efficient creation process, the second step ensures that complex dependencies, which cannot be discovered automatically, can be included into the model. It also enables users to refine discovered dependencies.

The discovery algorithm takes the process description of the composite service and determines all valid paths from the start node to the end node. Next, the services within each path are checked for horizontal dependencies. The underlying assumption for this process is that services, which do not occur within a path do not have consumer-provider based dependencies. Synchronization constraints can occur also across paths, but they would have to be expressed explicitly since neither process description nor SLAs contain this information implicitly. Vertical dependencies are discovered by comparing the single services inside a path with the composite service. Dependencies regarding the QoS and price of services are not analyzed based on the created paths, but instead require a precise analysis of the process structure. QoS and price dependencies occur as 1-to-n relationships between the composite service and the atomic services. These dependencies are expressed as a function for calculating the respective composite value from the atomic values. The formula for composite value calculation is generated based on the process structure [13].

5 Models for the Internet of Services

The intrinsic complexity of IoS-based services requests for a new approach in Service Engineering (SE) and tools in developing such services [14]. Typically, services evolve in a common ecosystem in which organizations and IT provide value in form of services. SE provides methodologies to cope with the complexity of several business actors and their interaction. Furthermore, SE specifies tools for implementing and deploying services, covering both, IT and business perspectives.

Consequently, SE is a structured approach for creating a new service. It addresses two problems: 1) multiple stakeholders across different organizations and 2) different perspectives ranging from business to IT. To cope with these challenges we propose an integrated service engineering methodology and support by meta-model and model matching.

5.1 Integrated Service Engineering

For the development of IoS-based services we proposed the Integrated Service Engineering (ISE) methodology [14] and implemented it in the ISE workbench [15].

Thereby, we present a model-based approach; i. e. each aspect of a service is formally represented by a corresponding model. Figure 4 shows the ISE framework which is part of ISE methodology. Inspired by the Zachman framework and following the separation of concerns paradigm, it structures services into four main perspectives and five dimensions. These dimensions are: service description, workflow, data, people and rules. Each of these dimensions is divided into four perspectives (layers) of abstraction. These perspectives of the ISE methodology can be regarded as a phase in the development (model refinement) of services. Thus, the models which are assigned to each layer support the development from different viewpoints (i.e., scope, business, logical, and technical).

Additionally, models at different dimension but belonging to the same layer are bound to others in order to form the complete business service model at the respective level of abstraction. For all cells of the matrix, we have defined formal models which should be considered in the service development. Examples of models include UML, BPMN, BPEL, OWL, etc. Figure 4 presents the models selected and used in the ISE workbench.

Therefore, ISE acts as an integration platform for several models placed in cells of the framework. Throughout one dimension, models are created with respect to different views and refined until they conform to a technical specification. This leads to multiple representations of information on different layers of abstraction in the corresponding dimensions. Changes in one model have to be propagated (or at least detected) into related models holding overlapping information (depicted by arrows in Figure 4).

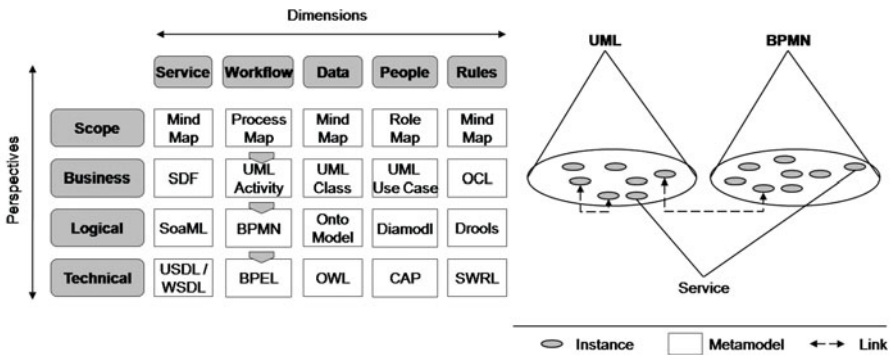


Fig. 4. The ISE models arranged in the corresponding matrix

5.2 Support by Model Matching

Multiple stakeholders and multiple perspectives result in several models designed in different ways. These models exhibit commonalities which need to be synchronized. This requires an integration of these models. The integration challenge is twofold: (1) one has to integrate the models by means of model transformation enabling an automatic synchronisation and (2) if a transformation is not available, one needs to identify commonalities. This covers use-cases such as duplicate detection as well as trace link creation. Thereby, having the trace link data available, common scenarios like change impact and orphaned elements analysis can be performed.

Figure 4 depicts a service, which is constituted of several models, each of them corresponding to a meta-model, i. e. a formalism representing a number of instances, e. g. a UML-class diagram or a BPMN process. These instances (models) have common features, since they represent the same services. For instance, a data object used in the BPMN has been modelled in a UML-class diagram. This information needs to be kept in sync in order to fulfill consistency. Nowadays, this is poorly supported by tools and if at all performed manually.

However, our approach has one major advantage; it is model-driven, which allows for a common formalism and therefore easy access to data in a closed world. To tackle the integration issue, we proposed to extend and adapt the aforementioned schema matching service, enabling it to consume meta-models [16]. We envision an extension allowing also a consumption of models, so correspondences can be discovered on meta-model *and* model level, thus performing meta-model *and* model matching in one system. This allows for a matching of BPMN and BPEL as well as BPMN and UML or any other meta-model. But we are not limited to these meta-models, but also support the reoccurring task of matching of concrete instances such as BPEL processes or Java classes. We name this approach *Layer Independent Matching*, since it is applicable to the meta and instance layer. Finally, one can match heterogeneous specifications, thus discover similarity (trace links) between different models like BPMN, BPEL, WSDL, USDL, etc as well as their concrete instances.

In model matching (instances) a bigger set of data is available compared to meta-model matching, so we feel that a stronger focus on structural heuristics is needed. Following that, we propose to apply graph edit distance algorithm taking advantage of planar graphs and using different clustering algorithms to cope with the increased dimension in size of models. For instance, a comparison between two concrete BPEL processes often contains more than 200 elements. Assuming they are represented as formal models in a graph this can be extended (e.g. in Java classes) to more than 5000 nodes, comparing 5000 x 5000 nodes leads to 2.5 Mio nodes which requests for a clustering approach, thus reducing the dimensions of the problem to be matched.

6 Conclusions

In order for the Internet of Services to become a reality, numerous areas of research need to be (re)explored. From business science, contributions on new business models and pricing schema will be valuable. In the area of law and cyberlaw, new legal matters related to the provision and contracting aspects of IoS-based services supported by networked information devices and technologies will be required. From the area of social science, new community rating schema will be needed. The spectrum of research topics is substantial and sizable. In this paper we have centered our attention on four main topics: the notion and characteristics of IoS-based services, the characterization of platform services, the management of SLA contracts, and the design of complex IoS-based services. To correctly understand the notion of IoS-based services, an historical retrospective allied with a detailed identification of the specificities of day-to-day services that can be digitalized into the IoS are fundamental. The next topic presented platform services and introduced a taxonomy to better understand the type of platform services

provided by marketplaces and provisioning platforms. Understanding the shared value-added contribution of an IoS-based service and the contribution of platform services is important to identify the focus of innovation and uniqueness. The third topic of study was the management of dependencies between services in compositions. We described an approach for the handling of dependencies at design and runtime. At its core it has a dependency model which is created by a semi-automatic approach of automatic discovery and additional modeling. Finally, the fourth topic described a structured and model-based approach to design and handle the intrinsic complexity of IoS-based services. Once individual models to describe a service are obtained, the challenge is to integrate the models using model matching and transformation. We presented a solution for supporting a semi-automatic matching of metamodels, models and instances using a Layer Independent Matching approach.

Acknowledgements. The TEXO project was funded by means of the German Federal Ministry of Economy and Technology under the promotional reference 01MQ07012. The authors take the responsibility for the contents. The information in this document is proprietary to the following Theseus Texo consortium members: SAP AG, empolis GmbH, intelligent views gmbh, ontoprise GmbH, Siemens AG, Fraunhofer Gesellschaft, FZI Forschungszentrum Informatik Karlsruhe, the German Research Center for Artificial Intelligence (DFKI GmbH), Technische Universität Darmstadt, Technische Universität Dresden, Technische Universität München and Universität Karlsruhe (TH). The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law.

References

1. Stafford, T.F., Saunders, C.S.: Moving towards chapter three. *e-Service Journal* 3(1), 3–5 (2003)
2. Nelson, R.R.: Technology and global industry: Companies and nations in the world economy. In: Guile, B.R., Brooks, H. (eds.), (1987) vol. 18, National Academy Press, Washington (April 1989)
3. Baida, Z., Gordijn, J., Omelayenko, B., Akkermans, H.: A shared service terminology for online service provisioning. In: Janssen, M., Sol, H.G., Wagenaar, R.W. (eds.) *Proceedings of the Sixth International Conference on Electronic Commerce ICEC 2004*, ACM Press, New York (2004)
4. Hull, R., Benedikt, M., Christophides, V., Su, J.: E-services: a look behind the curtain. In: *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 1–14. ACM Press, New York (2003)
5. Long, J.: *ITIL Version 3 at a Glance: Information Quick Reference*. Springer Publishing Company, Incorporated, Heidelberg (2008)
6. Kotler, P.: *Marketing Professional Services*. Prentice-Hall, Englewood Cliffs (2002)
7. Payne, A.: *Essence Services Marketing*. Pearson Education, Limited, London (1993)
8. EU directive 2006/123/EC of the European parliament and of the council of December 12 2006 on services in the internal market. Technical report, European Union (2004)

9. Economic Assessment of the Barriers for the Internal Market for Services. Technical report, Copenhagen Economic (2005)
10. Winkler, M., Schill, A.: Towards dependency management in service compositions. In: Proceedings of the International Conference on e-Business, pp. 79–84 (2009)
11. Malone, T.W., Crowston, K.: The interdisciplinary study of coordination. *ACM Computing Surveys* 26(1), 87–119 (1994)
12. Winkler, M., Sell, C., Springer, T., Schill, A.: A dependency model for composite service management. In: Proceedings of the International Conference WWW/Internet 2009 (2009) (to appear)
13. Cardoso, J., Miller, J., Sheth, A., Arnold, J.: Quality of service for workflows and web service processes. *Journal of Web Semantics* 1, 281–308 (2004)
14. Cardoso, J., Voigt, K., Winkler, M.: Service engineering for the internet of services. In: Enterprise Information Systems, Springer, Heidelberg (2008)
15. Scheithauer, G., Voigt, K., Bicer, V., Heinrich, M., Strunk, A., Winkler, M.: Integrated service engineering workbench: Service engineering for digital ecosystems. In: International ACM Conference on Management of Emergent Digital EcoSystems (2009)
16. Voigt, K.: Towards combining model matchers for transformation development. In: Proceedings of 1st International Workshop on Future Trends of Model-Driven Development at ICEIS 2009 (2009)

Pragmatic Web Services: A Semiotic Viewpoint

Kecheng Liu and Adrian Benfell

Informatics Research Centre, University of Reading, Ground Floor
Building 42, Whiteknights, Reading RG6 6WB, U.K.
k.liu@henley.reading.ac.uk, a.j.benfell@reading.ac.uk

Abstract. For an organization to discover and consume contextually relevant web services over a Service Oriented Architecture (SOA) requires competencies that move beyond the current boundaries of Syntactic and Semantic Web technology. The theory behind the Pragmatic Web (the study of intentional effects via communicative acts using the web) and Semiotics (the science of signs) can make a significant contribution to formulating an approach to discovering and consuming web services in the format of Pragmatic Web Services. Pragmatic Web Services as defined in this paper affords an array of opportunities should an organization wish to implement a SOA based upon web services whilst taking into account any related contextualized organizational parameters.

Keywords: Pragmatic, Semiotic, Semiosis, Service oriented architecture, Web services.

1 Introduction

Web technology allows organizations acting as service providers or consumers to work with web services whilst cooperating together across spatial and temporal differences. Organizations work within a variety of social parameters that can have a significant influence on the discovery of appropriate web services. Papazoglou [23] says a major test linked to web service discovery relates to detailing organizational objectives that correlate with a class of desirable web services required using some kind of formal request language. Other authors also agree with this position [2], [10], [15] and [24]. Additionally, these authors focus upon the obstructions linked to web service discovery, as they assert that web service discovery from an organizational perspective is the weak component of the tri-themed web service model (service provider, service consumer and service broker). Consensus emerges between these authors to suggest that web service discovery is difficult to address within different organizational contexts. Furthermore, problems can occur when an organization crosses its boundary in search of web services; the question of matching web services to organizational parameters becomes even more demanding. However, the intrinsic features of the Pragmatic Web as the basis for communicative acts between service providers and consumers and Semiotics for understanding the signs exchanged during communication, are both amenable to the challenges of discovering web services in diverse organizational settings. This paper outlines SOA and web services and a dominant SOA framework, the OASIS Reference Architecture [20] as they relate to

the social implications of realizing a SOA. The theory supporting the Pragmatic Web and Semiotics is then used to show how web services can be discovered as ‘Pragmatic Web Services’ within a SOA.

2 Service Oriented Architecture and Web Services – A Motivating Example

Service Oriented Architecture (SOA) provides a rigorous model for capitalizing upon the claimed benefits of web services. While the service-oriented approach provides an effective solution to using various web services to support business processes, it involves a number of key factors that critically determine success. Organizations encompass social parameters that are defined by strategic objectives, organizational structures, people, and business processes [4], [7] and [20]. The social parameters of business organizations inform normative behavioral patterns established by people interacting within their domain. The matching of web services using SOA as the platform to represent the social parameters of individual organizations is beyond the capability and scope of the current design and implementation of web services and SOA research [23]. Many current effective technical solutions are still not capable of coping with complex challenges that emerge from organizations that function from within their set social parameters. The interface of a web service follows a syntactic structure organized in a service catalogue, in line with a stack of web service protocols allowing them to be called to deliver their functions across spatial and temporal differences when required. According to the capabilities of each web service, a web service can perform functions synchronously or asynchronously dependent upon a specified communication protocol [12]. The communication protocols supporting web services enable services to be requested and delivered over a SOA platform to execute the necessary functions to fulfill business processes [14]. However, the interface belonging to a web service imposes a software implementation on a service consumer; the interface is based upon a narrow functional description not configurable to the social parameters of an organization.

To highlight these complexities, the OASIS Reference Architecture [20] is used as a prototype to illustrate the need for further examination from wider perspectives. For example, the OASIS Reference Architecture stresses the importance of considering the social parameters aligned to organizations when implementing a SOA based upon web services:

“The mode of business in a SOA based system is characterized in terms of providing services and consuming services to realize mutually desirable real world effects. The people and organizations involved in a SOA based community may be a single enterprise or a large scale peer-to-peer network of enterprises and individuals. Many of the activities that people engage in are themselves defined by the relationships between people and organizations that they belong to.” [20, p22].

The OASIS Reference Architecture stresses that executing a SOA typically results in a change to the ‘social structure’ of an organization [20, p24]. Within a social structure, ‘social norms’ emerge as an assortment of constituents, for example people, software agents and various other SOA resources of the social structure interact. A component described as the Business via Services View of the OASIS Reference

Architecture illustrates a generic social structure of an organization implementing a SOA. The Business via Services View is composed of a variety of other models to describe the stakeholders and participants, the resources they have access to and their needs and capabilities. The stakeholders and participants are described as the foremost agents belonging to a SOA. It includes a number of definitions, but generally the terms used revolve around the functions of a service provider and consumer as ‘participants’ in SOA. The core of this paper is based upon the social structure and resultant joint interactions between a service provider and consumer (as groups of people) that implementing a SOA may define. According to [20] a social structure describes emergent rules that are generally recognized by all participants. A ‘constitution’ within the OASIS Reference Architecture defines the roles of participants within the rule based framework and their obligations based upon those agreed rules. A ‘shared state’ emerges when a set of ‘social actions’ (actions in social structure) take place leading to social facts (social facts that are understood by a social structure). A social fact is validated as a ‘social norm’ when the social structure attaches meaning to the social fact. During a process of validation, normative behavioral patterns emerge that all participants agree to. A ‘commitment’ is a social fact that can be true (or false should the commitment not be adhered to) when, in a future state, a participant is obliged to carry out a social fact. The OASIS Reference Architecture marks out how the social parameters of organizations are formulated when implementing a SOA (with web services) based upon its description of a social structure. It also provides a channel to add the concepts drawn from the Pragmatic Web and Semiotics to define what Pragmatic Web Services are and how they affect the discovery process.

3 The Pragmatic Web

The ‘web’ as it relates to web services is often referred to in three formats: the Syntactic Web – predominantly used to present the interfaces of web services; the Semantic Web – using data to generate more meaning related to the orchestration of web services to suit business processes; and the Pragmatic Web – the contextualization of web services through the exchange of signs based upon communicative acts. Using the web as a tool to access web services so that they can be embedded within software applications is a significant development of the Syntactic Web. However, the limitations of the Syntactic Web surface when considering web services. For example, the characteristics of web services are described using a Web Service Description Language (WSDL) document based upon the Extensible Markup Language (XML). The WSDL defines the web service elements such as the data types, parameters, operations, and the binding and communication stack of protocols, so that a service consumer can make use of internet and web technology to operate any web service [8] and [16].

The elements of a WSDL file are an essential ingredient to join a service provider and consumer in a business transaction. However, the syntactic mechanisms that make web services work on the Syntactic Web do not provide the facilities for a service provider to describe their web services effectively, and conversely a service consumer to make informed choices. Furthermore, joint actions as they are described within the OASIS Reference Architecture assume that service providers and consumers carry out their work independently, and join forcefully only when a web service is called for using the WSDL to coordinate their interaction.

The Semantic Web by contrast, Berners-Lee et al [5], is based upon the idea that meaning can be obtained from the XML mark-up to provide more understanding related to individual web services, Shadbolt et al [28]. The discovery of web services as it features within the realm of the Semantic Web is to use software-based agents to ensure that relevant web services are the ones selected on behalf of a service consumer only. To achieve semantic web service discovery, ‘static ontologies’ are applied to construct a domain of interest that resides within an organization, Schoop et al [29]. The ontological units described are typically objects (or instances), classes, attributes (or slots) and relations. Objects are individual instances of a class, and a class describes a collection of similar objects. Prime examples of such ontology languages include the Web Ontology Language (OWL) [21] and one specific to web service discovery OWL-S [22]. Ontology languages permit a decentralized approach to explain in more detail the salient features of web services. However, this category of ontology modeling is a weakness as they facilitate the modeling of data structures only, suggesting they have limitations when the social parameters of organizations must be considered. This places a limit to the effectiveness of capitalizing upon OWL and OWL-S to assess web services from organizational perspectives.

Stamper [31] recognizes three categories of ontology modeling that provide clues to their usefulness in connection to web service discovery. The first relates to the recognition of symbols as they are typically found in any standard presentation format, for example the Syntactic Web. The second identifies distinct objects and object type classification, similar the type featured in OWL/OWL-S as used within the Semantic Web. The third type of ontology is based upon the view that the world known to a person consists of only the actions a person can carry out in their environment. To overcome the limitations of the ontologies that can be modeled using languages such as OWL/OWL-S within the setting of web service discovery, Stamper’s [31] third type of ontology is particularly relevant and conclusively finds its way into the Pragmatic Web.

Singh [30] put forward an opinion about some issues that could also affect the take-up of the Semantic Web – chiefly related to the ontology modeling tools used. Schoop et al [29] provides support to Singh’s outlook on the Semantic Web by saying “the most problematic assumption (belonging to the representation of ontologies within the Semantic Web) is that context-free facts and logical rules would be sufficient”. Singh [30] recommends the Pragmatic Web as a companion to the Semantic Web to resolve the issues that could impair the take-up of the Semantic Web. The exposure made by Stamper [31] related to ontologies is further emphasized by deMoor [11]. The three principles of the Pragmatic Web made by Singh [30], who builds his work on [19], is revised further by Liu [18] as:

- User before provider: the emphasis of the Pragmatic Web focuses upon the effect of information about web services on service consumers, rather than on the intended meanings supplied by the service providers about web services as defined in a static object type ontology;
- Process before data: the execution of the application, the execution of a web service is important, which constitutes a ‘run-time’ context. Such a context will determine the meaning dynamically; therefore the Pragmatic Web is context-aware;

- Interaction before presentation: dynamic interpretation of web service suitability produces the actual effect of communication; the meaning generated in the process of interpretation cannot be dependent on the presentation of a static object based ontology that is found within the Semantic Web.

Pragmatics as it features within the Pragmatic Web is a scientific discipline to study intention and effect of human communication. Syntactics, semantics, and pragmatics as they correlate to the three versions of the web, also constitute the traditional discipline of studying signs used in communication and information processing – Semiotics. Furthermore, semiotics puts the evolution of the web on a sound footing as it changes the focus from syntactics and semantics to pragmatics, thus reflecting the advancements of the Pragmatic Web [18]. The theory supporting the Pragmatic Web provides the conduit between a service provider and consumer to overcome the issue of matching web services to organizational parameters, particularly since the OASIS [20] framework advocates the ‘forceful joining’ of service providers and consumers using communication actions to underpin interaction. The framework in fig 1 affords the joining of service providers and consumers. Although derived from the OASIS Reference Architecture it takes into account that communicative actions are configured by intentional behavior in an interaction setting. In addition, the framework is used to shape how semiotics in the form of ‘shared semiosis’ can occur between a service provider and consumer. Shared semiosis is used to configure the joint actions between a service provider and consumer – a missing component from the OASIS Reference Architecture.

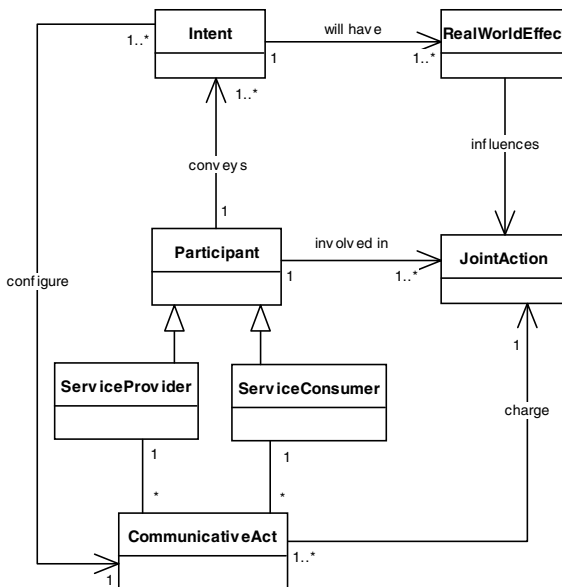


Fig. 1. Joint Action as Interaction

The provision and consumption of a web services are complex processes of shared semiosis carried out by service providers and consumers. For the provider, the semiosis takes place during the construction of a web service; while the semiosis for the consumer occurs when a web service is to be identified and used. Should the process of semiosis not be shared, the success of web service discovery relies upon the potential of a web service meeting a consumer request – a remote possibility should a service provider and consumer not communicate through joint actions as advocated in this paper.

To maximize the success of discovering suitable web services, the service provider and consumer must understand the semiosis of each participant in a joint action, and without any form of communication where there is an exchange of various signs, is highly unlikely. To overcome this issue, semiotics as the corroborating theory is used to put the concept of joint actions on a reliable foundation.

4 Semiotics as the Theoretical Underpinning

Semiotics is traditionally based upon two domains, Saussurean [26] linked to linguistic studies, and Peircean [25] where all kinds of signs including linguistic ones are investigated. Both domains adhere to a subjective element to understanding signs as signs can have different meanings across various social settings. Peirce's view of semiotics is used to arrange shared semiosis between service providers and consumers due to its triadic grounding. For instance, to form an understanding of a sign Peirce defined the term semiosis, the interplay between three unified parts: a sign, an object and an interpretant. A sign can take on various formats: symbolic, iconic and indexical. Symbolic signs identify their objects through cultural conventions such as words; iconic signs represent their objects by similarities, such as images; and indexical signs are signs that have causal links to meaning, for example a thermometer indicating a temperature, a petroleum gauge to indicate the level in a petroleum tank. Each sign classification has an order of 'agreed rules' associated with them to make them work. For example, indexical signs are considered to be highly subjective and symbolic signs such as words are not so subjective, as people will continually return and agree to their meaning.

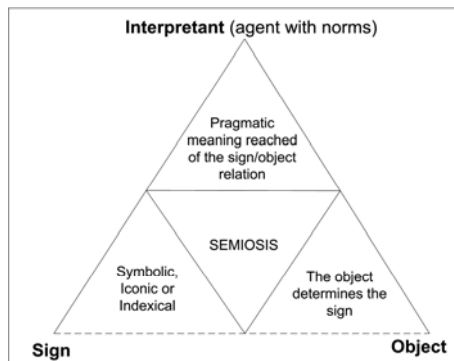


Fig. 2. Semiotic Triangle

Regarding the process of semiosis in fig 2 [17, p16], a sign refers to an object, and the object verifies its sign. The object is the semantic meaning of sign; there is an ontological dependency between the object and sign as the object confirms the existence of a sign. The sign – object relation is added to by the interpretant, consequently an interpretant is a more developed sign in terms of understanding. The interpretant holds the contextual meaning given to a sign – it can be made up of norms that provide the rules or law by which meanings belonging to an interpretant sign can be revealed. Furthermore, according to Liu [17] a semiosis is a sign-mediated process in which a person gives meaning to a sign, hence it is a way to build knowledge and experience attributable to people only. A semiosis therefore can only take place in a human communication setting as it is mediated using signs such as languages (either natural, computational or formal). Peirce [25] also introduced three modes of inference that sit within the process of semiosis, abduction, deduction and induction. The purpose of abduction is to develop a hypothesis to explain the observations made related to a sign. Deduction generates an incomplete conclusion related to the meaning of sign, the object, but confers the existence of a sign. Induction completes the conclusion of a sign by verification through consensus. To complete a full understanding of an inquiry through semiosis, abduction, deduction and induction must be used in combination to infer the meaning of a sign. Induction as a part of semiosis in a shared format draws in the roles of service providers and consumers in joint actions that can be described by affordances, norms and deontic logic [17].

4.1 Affordances, Norms and Deontic Logic

The theory of affordances originates from Gibson [13] and was extended to the study of the real world for understanding patterns of human behavior, Liu [17]. Society as an environment sets boundaries that enable many patterns of behavior possible. Should a person (a service provider or consumer) be removed from its environment, the range of behavior the person owns would cease to exist. Also, according to Liu [17], a person by knowing the invariants (general patterns of behavior) can learn the acceptable social boundaries. The OASIS Resource Architecture refer to these as ‘shared states’ that emerge when a set of ‘social actions’ (actions in a social structure) take place leading to social facts (social facts that are understood by a social structure). A social fact is then validated as a ‘social norm’ when a social structure attaches meaning to the social fact, affordances in this context subsume norms that fall into two categories, social and legal.

According to Boella et al [6] social norms “exist where models of human activity are normatively regulated”. Essentially social norms provide a mechanism by which people through consensus, recognize and adhere to. Social norms not only determine the meanings people attach to signs but also introduce unpredictable behavior [32]. Regarding norm enforcement, social norms are ‘enforced’ by coordination motives that are governed by social phenomena as they are shared expectations about a solution to a given problem, hence there is no need for rigid enforcement.

Social norms are governed by expected patterns of behavior; ‘legal’ norms however require a different form of control. Legal norms are sustained by social disapproval or a form of punishment for norm violation. Legal based norms are an expression of obligations and rights that a person has within a social structure of some kind [6] and [17]. Typically legal norms are supported by a consenting mechanism

that is accepted by people since they can be used to overcome problems of coordination and cooperation. The purpose of considering social and legal norms in a shared semiosis between a service provider and a consumer is to initiate a mechanism whereby norms are created and agreed upon by both parties; they are used to describe the elements of web services as ‘codes’.

Deontic logic includes obligation, permission and forbiddance. Social and legal norms can be specified using these deontic operators as they fall into the category of behavioral norms – that is what people obligatory (must), permitted (may) and forbidden (must not do) [17]. The list of deontic operators relates to the notion of emergent social norms produced when the various parts of a social structure cooperate in a SOA. Legal norms are imposed on organizations through acts agreed by society, such as government. The deontic operators are therefore used to capture the social and legal norms that explicate the social parameters of an organization, conferred by affordances, the repertoire of behavior known only to people acting as in a service provider or consumer organization. Hence affordances represent the ‘business knowledge’ that is used in the web service matching process in a shared semiosis.

4.2 Speech Act Theory

An ‘illocutionary act’, described by Austin [1], models the intention that a speaker makes by saying something to a hearer, and the hearer doing something. Austin establishes two further ideas that supplement the illocutionary act, constative utterances (true and false) and performative utterances (to do something by speaking). Austin goes further by saying that an illocutionary act consists of ‘securing the uptake’, that is making sure the act is executed. To strengthen an illocutionary act, Searle [27] embellishes Austin’s theory by adding illocutionary points, for example:

- Assertive – to commit the speaker (in varying degrees) to something being the case, or to the truth of the expressed proposition;
- Directive – to attempt (in varying degrees) to request the hearer to do something. These include both questions (which can direct the hearer to make an assertive speech act in response) and commands (which direct the hearer to carry out some linguistic or non-linguistic act);
- Commissive: to commit the speaker (in varying degrees) to promise to do some future course of action;
- Declaration: to bring about the correspondence between the propositional content of the speech act and reality (e.g., pronouncing a couple married);
- Expressive: to express a psychological state about a state of affairs (e.g., apologizing and praising).

The components of Speech Act Theory (SAT) are used to generate communicative acts to support joint actions between service providers and consumers; they interface the exchanges between service providers and consumers as joint actions. Table 1 depicts where the Pragmatic Web and Semiotics can be used to fortify how joint actions are carried out in SOA and web service discovery setting.

Table 1. SOA, Pragmatic Web and Semiotics

SOA and Web Services	Pragmatic Web	Semiotics
Communication Actions.	Speech Act Theory.	Speech Act Theory.
Joint Actions.	Effect of information through communicative acts.	Shared semiosis configures the communicative acts.
Real-world effects (Intent).	Context awareness through communicative acts.	Initiated by shared semiosis.
Obligation.		Deontic operators.
Social Norms.		Affordances and norms (Social and Legal) used as a part of shared semiosis.

To summarize, the relationship between the Pragmatic Web and Semiotics is particularly relevant to the study of web services in service oriented contexts. For example, the shared semiosis approach to joint actions facilitated by communication acts is the starting point in defining Pragmatic Web Services.

5 Embedding Semiotics into Web Service Discovery

Within service oriented contexts, the signs generated by a service provider and consumer are inclined to be textually orientated, for example a WSDL file. A service provider by authoring such a text goes through a process of semiosis. Equally, a service consumer by examining a WSDL file to assess the functional properties and capabilities of a web service also completes a process of semiosis. To place shared semiosis within the context of SOA and web services, Liu [17] describes four parts that are applicable:

1. **Universal** – it explains the process used for creating and using a sign (the codes created and used in a text describing the elements of a web service as communicated between a service provider and consumer);
2. **Specific Criterion or Norm** – semiosis is a process capable of identifying anything present according to the study of signs (it relates to the dependencies between the codes describing the elements of a web service that a service provider and consumer agree to be present);
3. **Subject-dependent** – it is closely related to the interpreter who can be an individual, or a group for example, with certain knowledge obeying certain affordances and norms (the interpretant in Peircean semiotics as it relates to the affordances and norms owned by the service providers and consumers that influence the inferences made regarding the codes used to describe the elements of a web service);
4. **Recursive** (an object or an interpretant generated by a service provider and consumer may become a different sign to be interpreted in a temporal joint action).

Additionally, syntactics as a branch of semiotics considers the formal properties of symbols as they relate to how words are used to form phrases and sentences inside texts. Syntactics provides a route into determining how a shared semiosis starts between a service provider and consumer. Understanding the syntactics of a text

through a process of shared semiosis, works as the starting point in a dynamic (temporal) series of events as syntactics, semantics and pragmatics are used in harmony to produce shared meanings. A service provider and consumer, for example, would review a text that contains various codes (syntactics) and derive a meaning (semantics) of each code. A full understanding of web service potentiality can only be reached when the effects of the sign, made by contextual interpretation, is made by the service provider and consumer through consensus (pragmatics). Based upon syntactics, semantics and pragmatics, and the four elements of semiosis put forward by [17], the basis for shared semiosis that frames the evolution of Pragmatic Web Services is defined as:

- Syntactics: an agreement made between a service provider and consumer relating to the structure and expected content as a universal (elements) of web service description files;
- Semantics: dynamic recursive agreement relating to the meaning of elements through coding contained within web service description files;
- Pragmatics: subject-dependent interpretation of the applicability of web services using a specific criterion or norm linked to affordances.

With reference to fig 3 (and arrow 1) below a range of textual documents conveying signs based upon ‘intertextuality’ influence the encoding of description specific to web services (abduction). Intertextuality, described by Chandler [9] refers to the authorship of a text and its encoding being driven by relationships to other texts. For example, a WSDL file is linked to other files such as a requirement specification generated during software development. A person constructing a web service description file would interpret the text in a specification document through the process of semiosis and accordingly document a web service with descriptions. Encoding therefore relates to the production of a textual document containing the codes that service providers and consumers agree to as a shared vocabulary. Each code in an XML format is related to the elements found in a WSDL file, but each code must also have representation in a norm and consequently as an affordance.

Textual encoding of documentation, shown in fig 3 (arrow 1), associated with web service description in combination with intertextuality, complements authorship with other available texts, and is fundamental when a shared semiosis takes place between a service provider and consumer. Table 2 below depicts the process of semiosis as it is aligned to a service provider and consumer linked to any available texts. A service consumer would encode a web service requirements specification that reflects the social parameters of their organization embodied as functional requirements – exposed as affordances and norms. Table 2 also treats service provider and consumer semiosis not as an independent activity prior to the publication and search for web services as the codes used to define web service elements must be agreed first through joint actions. A service provider or consumer decodes a text (deduction) through shared semiosis by following the agreed codes. A service consumer (as a reader) would participate actively in constructing a meaning associated with a textual document [9]. Applying the term ‘denotative sign’ (arrow 2 in fig 3) for textual documentation a service provider and consumer would arrive at a shared understanding (but incomplete of its full interpretation) of any collective text by following the agreed

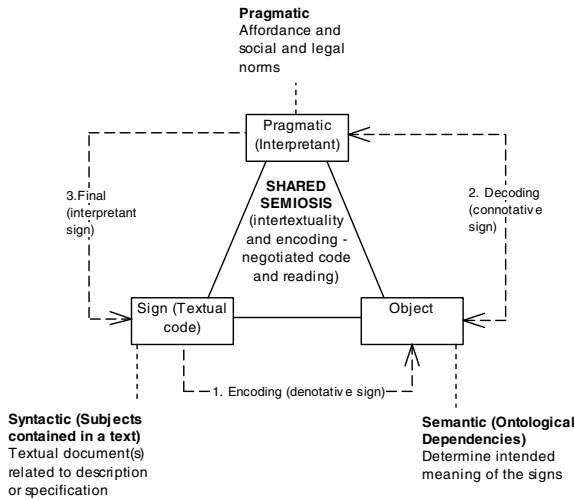


Fig. 3. Shared Semiosis

Table 2. Semiotic Branch

Semiotic branch	Intent and Real World Effect	Semiosis (intertextuality – encoding, negotiated code and reading)
Syntactic Encoding (denotative sign).	Capture through existing texts the elements to form codes that describe the syntactic features of a web service.	Sign – Textual code (intertextuality and encoding) Source code, analysis and design specifications – narrative and diagrammatic models.
Semantic Decoding (connotative sign).	Comprehension by consensus (using communicative acts) the elements symbolized as codes in relation to the functions and capabilities of a web service.	Object – Connotative sign (negotiated code and reading) Ontological dependencies linked to the contextualized interpretation by an Interpreter (Participant).
Pragmatic (interpretant sign).	Linking the interpretations of the elements with potential contexts and effects on all participants and specifying a meaning of all subjects congruent with all participants.	Interpretant – Connotative signs linked to the social parameters of a business organization defined as affordances and structured using norms.

codes. Hence, a service provider or consumer can ascertain the primary meaning conveyed through consensus, or as in the case of a web service, the elements of a web service represented as codes. The denotative sign (or signs) defining an agreed vocabulary between a service provider and consumer opens up further interpretation of connotative signs (induction).

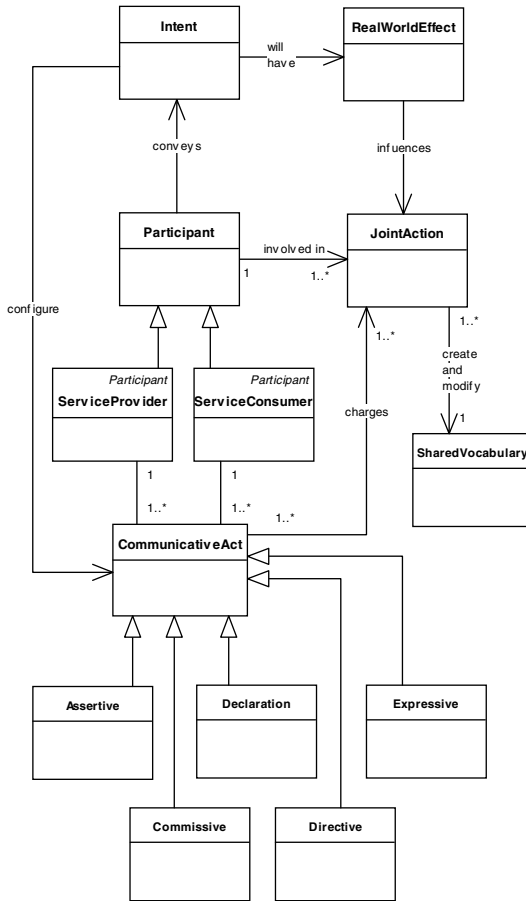


Fig. 4. Pragmatic Web Services

The decoding of a sign initiates a denotative sign, a consensus (objective) view of the sign (code) meaning. However, as a service provider or consumer decodes a joint vocabulary, any social parameters and ‘personalized’ associations linked to the codes influence the meaning. Hence, once an agreement is formed through denotation a service consumer will influence the process of shared semiosis through governed social parameters and independent understanding of the codes created by the service provider. Consequently, connotative signs emerge that emphasize the need for a deeper shared semiosis. Negotiated code and reading relates to the process that a service provider or consumer will resist and modify meaning of a text according to social position, personal experiences and interests through affordances but will generally accept the preferred meaning intentionally communicated. Therefore, a final interpretant sign based upon Peirce’s view of induction [25] must be reached by a service consumer following instances of shared semiosis at different time intervals.

Not until an interpretant sign is reached relating to each sign (code) that a complete vocabulary of code definitions is available. In essence, the social properties belonging

to an organization inform the interpretation of signs – the interpretant is then linked with the meaning of codes in a text. The agreed shared vocabulary completes a ‘first level’ shared semiosis between a service provider and consumer.

The service consumer moves to a diagnostic and prognostic phase to determine the final interpretant sign. During diagnosis the service consumer would identify the suitability of a web service by linking the meaning obtained through shared semiosis (vocabulary) with organizational context. However, the shared vocabulary as agreed through shared semiosis is temporal, as the service provider and consumer may return to a shared semiosis as a result of diagnosis. Prognosis where the final interpretant signs exist determine the acceptance or refusal of a web service – should agreement not be made pertaining to the codes used by a service provider or consumer, each is obliged (an obligation state in the OASIS Reference Architecture) not to continue in a joint action. Hence, the pragmatic matching of web services is made explicit through an agreed shared vocabulary.

Joint actions as they are contingent upon the process of shared semiosis are informed by communicative acts [3]. Communicative acts convey intent as service provider and consumer behavior has an effect within the real-world (fig 4). The real-world effects change shared states (shared states as described in the OASIS Reference Architecture). All of the components in fig 4 solidify the definition of Pragmatic Web Services. Furthermore, key to this definition is the notion of shared semiosis as it harmonizes each component of the framework.

6 Conclusions

The emergence of the Pragmatic Web opens up a frontier to capitalize upon web service technology. However, the challenges are far from technical, and many are beyond the capability of the current approaches. The Pragmatic Web and Semiotics offers a sound theoretical basis for emergent research into SOA and web services. The semiotic perspective offers a useful insight in the theoretical underpinning for successful web service discovery in the service oriented paradigm. The approach to web service discovery described here illustrates how the role of shared semiosis structured as joint actions in the provision and consumption of web services can work.

For example, the purpose of capturing a communicative act and applying semiosis to the symbolic signs (the codes) generated by joint actions can provide a means to capture intentions originating from service providers and consumers. The model in fig 5 shows that a service provider or a consumer communicative act can capture more intentions leading to infinite semiosis. Affordances are used to ensure that a request for a web service, for example, is compliant with a shared vocabulary.

The evolution of the Syntactic Web and the Semantic Web to the Pragmatic Web improves greatly the quality of the work related to searching for web services over a SOA platform. Semiosis as it emerges from Peircean semiotics is theoretically open to the challenges of matching web services to the social parameters set by organizations. The social parameters are known to the people who belong to an organization and are an essential ingredient that determine the success or failure of web service discovery. The social parameters are represented as affordances which are described by social and legal norms. Should an agreement not be made regarding the meaning of each

code, a service provider and consumer can each terminate a business transaction. Pragmatic Web Services are defined by the relationship between a service provider and consumer through the interaction of a shared semiosis orchestrated by communicative acts – joint actions to achieve the matching of appropriate web services. Should the concept of Pragmatic Web Services not be followed, the probability of matching web services is greatly reduced and left as a remote possibility.

References

1. Austin, J.: *How to Do Things with Words*. Harvard University Press, Cambridge (1962)
2. Atkinson, C., Bostan, P., Hummel, O., Stoll, D.A.: *Practical Approach to Web Service Discovery and Retrieval*. In: *IEEE International Conference on Web Services*, Salt Lake City UT, USA, July 9-13 (2007) ISBN: 0-7695-2924-0
3. Benfell, A., Liu, K.: *Defining a Pragmatic Based Web-service Discovery Mechanism, I-SEMANTICS Special Track: 4th AIS SigPrag*. In: *International Pragmatic Web Conference*. Graz, Austria, pp. 763–774 (2009) ISBN 978-3-85125-060-2
4. Berkem, B.: *From The Business Motivation Model (BMM) To Service Oriented Architecture (SOA)*. *Journal of Object Technology* 7(8), 57–70 (2008)
5. Berners-Lee, T., Hendler, J., Lassila, O.: *The Semantic Web*. *Scientific American*, 34–43 (May 2001)
6. Boella, G., Torre, L., Verhagen, H.: *Introduction to Normative Multiagent Systems*. *Computational and Mathematical Organizational Theory* 12(2-3), 71–79 (2006)
7. *Business Motivation Model – OMG*, <http://www.omg.org/spec/BMM/1.0/>
8. Carey, M.: *SOA What?*, pp. 92–93. *IEEE Computer*, Los Alamitos (March 2008)
9. Chandler, D.: *Semiotics – The Basics*. Routledge, New York (2002)
10. Crasso, M., Zunino, A., Campo, M.: *Easy web service discovery: A query-by-example approach*. *Science of Computer Programming* 71, 144–164 (2008)
11. de Moor, A.: *Patterns for the Pragmatic Web*. In: *Dau, F., Mugnier, M.-L., Stumme, G. (eds.) ICCS 2005*. LNCS (LNAI), vol. 3596, pp. 1–18. Springer, Heidelberg (2005) (invited paper)
12. Erl, T.: *SOA: Principles of Service Design*. Prentice-Hall, Englewood Cliffs (2008)
13. Gibson, J.: *The Theory of Affordances*. In: *Shaw, R., Bransford, J. (eds.) Perceiving, Acting, and Knowing* (1977) ISBN 0-470-99014-7
14. Huang, S., Chu, Y., Li, S., Yen, D.: *Enhancing conflict detecting mechanism for Web Services composition: A business process flow model transformation approach*. *Information and Software Technology* 50(11), 1069–1087 (2008)
15. Kim, I.-W., Lee, K.-H.: *Web Services, Describing Semantic Web Services: From UML to OWL-S ICWS*. In: *IEEE International Conference on Web Services*, vol. 9(13), pp. 529–536 (2007)
16. Li, Y., Liu, Y., Zhang, L., Li, G., Xie, B., Sun, S.: *An Exploratory Study of Web Services on the Internet*. In: *IEEE International Conference on Web Services*, Salt Lake City UT, USA, July 9-13 (2007) ISBN: 0-7695-2924-0
17. Liu, K.: *Semiotics in Information Systems Engineering*. Cambridge University Press, Cambridge (2000)
18. Liu, K.: *Pragmatic Computing: A Semiotic Perspective to Web Services*, Keynote paper. In: *Proceedings of ICETE 2007, E-Business and Telecommunications*, pp. 3–15. Springer, Heidelberg (2008) ISSN 1865-0929

19. Morris, C.W.: *Foundations of the Theory of Signs*. Chicago University Press, Chicago (1938)
20. OASIS Reference Architecture for Service Oriented Architecture Version 1.0, Public Review Draft 1 (April 23, 2008), <http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf>
21. OWL Web Ontology Language Overview, <http://www.w3.org/TR/2004/REC-owl-features-20040210/>
22. OWL-S OWL-S: Semantic Markup for Web Services, <http://www.w3.org/Submission/OWL-S>
23. Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F.: *Service-Oriented Computing: State of the Art and Research Challenges*, pp. 38–45. IEEE Computer, Los Alamitos (November 2007)
24. Pastore, S.: The service discovery methods issue: A web services UDDI specification framework integrated in a grid environment. *Journal of Network and Computer Applications* 31, 93–107 (2008)
25. Peirce, C.S., Hartshorne, C., Weiss, P., Burks, A.W. (eds.): *Collected writings*, vol. 8. Harvard University Press, Cambridge (1931)
26. de Saussure, F.: *Course in General Linguistics* (trans. Roy Harris), London, Duckworth (1916/1983)
27. Searle, R.: *Speech Acts*. Cambridge University Press, London (1969)
28. Shadbolt, N., Hall, W., Berners-Lee, T.: The Semantic Web Revisited. *IEEE Intelligent Systems*, 96–101 (May-June 2006)
29. Schoop, M., de Moor, A., Deitz, J.L.G.: The Pragmatic Web: A Manifesto. *Communications of the ACM* 49(5), 75–76 (2006)
30. Singh, M.P.: The Pragmatic Web. *IEEE Computing*, 4–5 (May-June, 2002)
31. Stamper, R.: Knowledge as Action: a Logic of Social Norms and Individual Affordances. In: Gilbert, G.N., Heath, C. (eds.) *Social Action and Artificial Intelligence*, pp. 172–191. Gower Press, Aldershot (1985)
32. Young, P.: Social Norms. In: Durlauf, S.N., Blume, L.E. (eds.) *The New Palgrave Dictionary of Economics*, 2nd edn., Macmillan, London (2008)

Part I

Enterprise Software Technology

Unified Hosting Techniques for the Internet of Tradeable and Executable Services

Josef Spillner, Iris Braun, and Alexander Schill

Technische Universität Dresden

Faculty of Computer Science, Chair for Computer Networks

Nthnitzer Str. 46, 01187 Dresden, Germany

{josef.spillner,iris.braun,alexander.schill}@tu-dresden.de

Abstract. In a collaborative Internet of Services, traditional hosting concepts from service-oriented architectures are not sufficient anymore. Instead of just registering service descriptions at central brokers, services are increasingly considered tangible, distributable and tradeable dynamic entities for all aspects around the service execution itself. The set of manageable service artifacts encompasses heterogeneous description files, usage contract templates and pricing models. Electronic services also contain implementation artifacts in a variety of formats. For service broker and marketplace operators, it is essential to be able to process these service artifacts. Each service technology usually requires a separate execution platform which eventually leads to a high complexity for the administration and management of services. Similar to the unification of invocation of web services through protocol adapters, it is clearly needed to unify management aspects like deployment, monitoring and adaptation of service implementations across technologies. We therefore propose the concept of unified hosting environments for tradeable and executable services.

1 Introduction

Software applications can run either locally on a user's computer or remotely on a server. Historically, each *remote application* required its own protocol with binary or text-based data transmission. More recently, the use of *extensible protocols* like HTTP and SOAP with a predefined structure and application-specific content expressed as XML, JSON or any other generic format has become popular especially in web-centric environments. In combination with a discoverable uniform invocation interface, the so-called service description, such applications are known as *web services*. The interest in web services is growing not only due to the technical merits. Instead, the more sophisticated web service management and trading facilities become, the more they will be used as electronic representation of conventional business services. From a business perspective, supplementing or replacing conventional services and processes with their electronic counterparts increases selection and composition flexibility, reliability and global availability.

Legacy software is often made accessible through the web for interaction with users, or as a web service for programmatic access. Such additions are known as *application service provisioning* (ASP) and lately as *software as a service* (SaaS). Despite this

evolution of interface additions, the server-side installation procedure and consecutive service management has not become easier. To date, there are no uniform administration interfaces available for managing heterogeneous services. Most approaches treat web services as pre-installed static software, essentially preventing the consideration of services as distributable entities.

On the other hand, a growing number of service offerings is being created with an *Internet of Services* [8] in mind. Easy installation, often just consisting of hot-deploying a self-containing *service package*, and registration at a *service directory* are among the characteristics of such services [12]. Due to the popularity of this development model, various different service execution techniques and programming languages as well as description and packaging formats for service deployment have been created and are widely used. Additionally, the runtime environments for such services range from operating-system level execution to highly sophisticated containers, execution engines and middleware infrastructures. In spite of the differences, they all provide a managing container for the deployed web service implementation. This assumption contrasts with recent research ideas for distributed, self-managed hosting [3] without containers, which is not yet widely used. We therefore refer to the individual middleware implementations of hosting environments for service execution and management as *containers*.

For service providers, this variety makes it difficult to offer the same level of hosting support and availability guarantees for any service. Even basic management tasks like retrieving a list of available services and service instances is not easily possible with today's systems. A unification of the containers into a *Unified Hosting Environment* (UHE) is thus needed. In this paper, we propose such a unification for the packaging, description and implementation of services and will show how heterogeneous service hosting in the Internet of Services becomes possible without excluding the existing service development community.

The remainder of the text is structured as follows: First, we present existing approaches for service trading and hosting and analyse their shortcomings. Then, we explain how to assess service packages and corresponding containers at an abstract level by representing their most important service hosting and management aspects in models. A formal container notation in UML based on these abstractions is given as the least common denominator of the unification concept. Third, we outline the concept of a UHE using this notation and well-known software design patterns.

2 Related Work

Background information on the challenges of introducing new network-level and application-level services into existing infrastructures is given in [11]. Both deployment and management, refined by us to cover monitoring and adaptation, has thus already been known to raise issues for about a decade.

A common issue with new technologies like SOA is the lack of widespread methodologies. The main contributing factors to this issue are heterogeneous stakeholders, programming languages, protocols, interfaces and implementation frameworks. The OASIS Framework for Web Service Implementation (FWSI) has been designed and

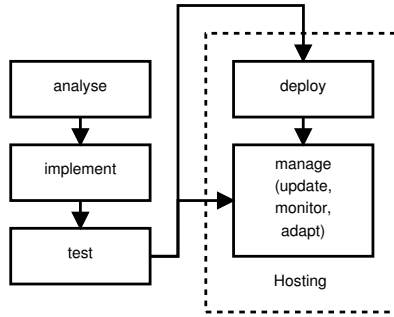


Fig. 1. Typical service hosting lifecycle

extended to cover the requirements analysis, development and testing of services, but the subsequent phases of deployment and operation are not yet covered [5]. The gap is shown in Fig. 1.

An approach to fill the gap is the Service Grid Infrastructure (SGI) [1]. Its purpose is the deployment and execution of heterogeneous services under a number of requirements: dynamic deployment, stateful services and consideration of non-functional properties. By restricting the service implementation technology to OSGi and EJB, the approach fails to take advantage of its otherwise sound decoupled management and execution of stateful services in a grid context. On the management layer, it offers unified access to monitoring data, but no unified adaptation interface. Despite an extension to .NET services later on [10], today's needs for dynamic service hosting range from small CGI scripts and BPEL files to complex, pre-configured virtual machines and corresponding SOA-aware management interfaces.

The Service Component Architecture (SCA) is an approach for unification of development and deployment of services. It wraps each service into a uniform component and defines dependencies between those [4]. However, there are several shortcomings with SCA regarding unified hosting. It requires the presence of the wrapper as part of the distributed service package. Furthermore, while a configuration interface is available, it does not cover dynamic reconfiguration at run-time. The omission of this capability leads to restrictions on adaptation mechanisms. Other dynamic aspects like switching connection protocols are supported, though. Therefore, we will not base our proposal on SCA but rather treat SCA as yet another concrete container in which SCA-compliant packages can be deployed.

Based on the evaluation of existing works, we formalise unified service hosting and propose a more complete and more suitable system which honours native service packages and hence doesn't require developers to learn additional technologies. Our proposal maintains a separation of concerns between access to services from clients and their management from hosting agents.

3 Abstraction of Service Packages

We propose the Tradeable Services Model (TSM) as light-weight, extensible packaging structure for services. TSM divides packages into declarative and imperative service

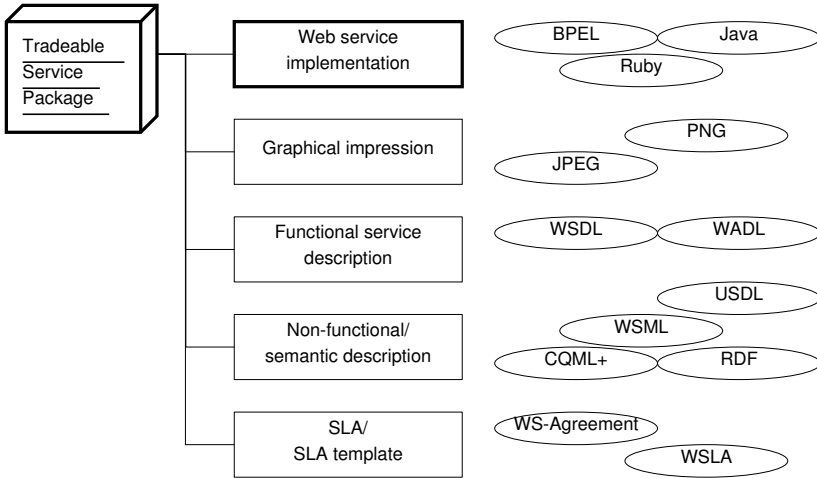


Fig. 2. Service package structure according to the Tradeable Services Model

artifacts. Declarative artifacts encompass the service descriptions in a variety of formats, contract templates, user interface hints and pricing models. The imperative artifacts only exist for executable electronic services for which they define the implementation. Some declarative artifacts are being used together with the imperative ones during service execution, such as syntactic service operation and parameter descriptions according to the Web Service and Web Application Description Language (WSDL, WADL). Others are purely of interest for service trading independently from the existence of a web service implementations, such as business and legal aspect descriptions according to the Unified Service Description Language (USDL). However, artifacts from the latter group might be bound to runtime-relevant artifacts including service level agreement templates, used to form access-controlling contracts during negotiation, and interpretable pricing models, used to generate bills based on monitored invocation patterns.

Service packages conforming to the TSM may be file archives, directories or other logical combinations of the artifacts. The package itself and each artifact shall be considered dynamic to allow for a flexible lifecycle management including insertions, updates and removals. The package structure along with typical representatives of each artifact group is shown in Fig. 2. This report will not elaborate on the abstraction of contents of declarative artifacts. Instead, the focus will be on deployment and management of the executable parts within service containers and related management of declarative artifacts as opaque entities.

4 Abstraction of Containers

The abstract notation of service execution containers shall be derived from a comparison of commonalities found in existing container implementations. Analogous to the TSM, we derive the Tradeable Services Container Model (TSCM) with a minimal set of management interfaces and propose to align container implementations to the model. The

Table 1. Overview on service containers

Container Overview				
	Container	Implementations	Deployment	Repository
1.	OSGi	Knopflerfish, Equinox	bundle, PAR	OBR
2.	Servlet	Jetty, Tomcat	servlet	none
3.	Web server	Apache, thttpd	LSB package	Debian
4.	OS	Linux		
5.	BPEL engine	ActiveBPEL, ODE	BPEL archive	none
6.	SCA runtime	Tuscany, Fabric3	SCA composite	none
7.	VM controller	Eucalyptus Cloud	Disk image	none
1) OSGi Bundle Repository, http://www.osgi.org/Repository/HomePage				
2) Debian Package Search, http://www.debian.org/distrib/packages				

Table 2. Dynamic aspects in service containers

Container Dynamics				
	Container	Configuration	Reconfiguration	Instantiation
1.	OSGi	manifest file	messages	singleton
2.	Servlet	web.xml	servlet reload	per call
3.	Web server	/etc config	server reload	per call
4.	OS	/etc config	SIGHUP	per call
5.	BPEL engine	none	none	per call
6.	SCA runtime	properties	none	various
7.	VM controller	/etc config	SIGHUP	singleton

variety of containers makes it non-trivial to find a practical least common denominator. Therefore, we selected typical representatives with a sufficiently high manageability and popularity in real-world deployments. This includes Java servlet containers, OSGi service platforms, conventional web servers for web applications, operating systems implementing the Linux Standard Base (LSB) service interface, BPEL engines and virtual machine controllers. Most of them expect services to be deployed in a format not understood by the other ones, and they also differ in their implementation languages, execution models and management interfaces. Table 1 shows a number of containers, implementations thereof and the accepted package formats for service deployment. For each package type, repositories are listed if they exist and are commonly used.

The capabilities of service packages differ widely. Automatic and semi-automatic functional testing, for example, requires a formal syntactical description of the service interface. In the case of BPEL archives, the corresponding WSDL files are generally included, whereas LSB packages don't contain such descriptions in most cases. A study conducted by us found that among more than 24,000 packages of the Debian GNU/Linux distribution, 7,121 are registered as being applications and 267 as being network services. These numbers are likely higher in reality since the process of

categorising the packages is ongoing. Yet, despite many of the service packages offering RPC, XML-RPC, SOAP or REST interfaces, only a total of two packages (*Sympa* and *phpWiki*) ship with WSDL descriptions.

More differences can be found in the runtime features of the containers. Some allow for a dynamic external reconfiguration or multiple instantiation, while others don't, as can be seen in table 2.

There is clearly a need to unify the access to such containers so that service hosters can cope with the heterogeneity imposed by the preferences of service developers.

5 A Formal Notation for Containers

Considering the variety of properties even of abstract views on containers, a symbolic expression will not be able to capture all aspects. On the other hand, extensible and machine-readable notations are hard to understand by humans and are unable to selectively focus on important details. Therefore, we use an UML notation to express a common model for service containers. An excerpt is shown in Fig. 3. The model refers to a single container which can deploy any number of packages. Each package contains n services which are each described by m formal service descriptions and l endpoints. For each service srv_i each invocation yields an instance $inst_{ij}$.

By extension, the invocation properties can be guaranteed with contracts sla_{ik} for each client for any service. Furthermore, specific service containers allow client-specific reservations and customisations of services which we call service allocations srv_{ik} . For instance, VM images are usually cloned and booted with individual configuration for resources like the number of CPU cores, available memory and virtual hard disk capacity. We will however leave discussion of handling contracts and service allocations out of this text for brevity.

Today's package formats support this model only to a certain degree. For example, the description and endpoint of a servlet-based service cannot be inferred from the servlet metadata automatically. Instead, heuristics need to be applied to find the

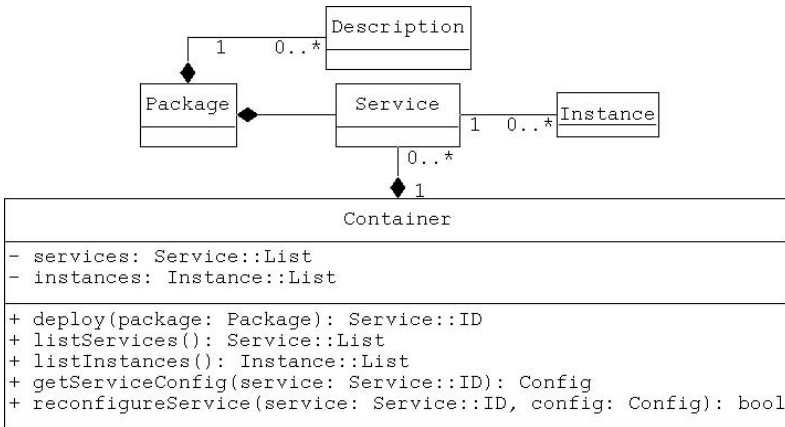


Fig. 3. Formal notation of service containers (excerpt)

correct information inside the package. OSGi-based and LSB services already provide rich metadata, including the licence which is important for the consideration of service migration. Still, the packaging formats could be improved to cover the needs of dynamically deployable services.

6 Unified Hosting Environment

Based on the abstract notation of containers, a collection of a number of them unified behind a delegating proxy yields a more powerful environment capable of handling more kinds of services.

For the deployment, a package containing the service is routed to the respective container. For example, a BPEL file is written to the BPEL server's hot-deployment directory, whereas a web application is stored on the disk followed by a notification of the web server. If the deployment at the container fails or no suitable container has been found, the UHE deployment fails.

Similarly, the reconfiguration either uses an appropriate mechanism of the respective container, or fails if either the mechanism fails or no container can be found. While the heterogeneous nature of containers and service concepts might pose a challenge to such combinations, preliminary findings suggest that at least among the considered implementations there are no hard discrepancies.

The architecture of the environment can thus be assumed to follow the pattern in Fig. 4, leading to Fig. 5. In terms of software design patterns, UHE is defined as a *proxy* connected to a number of *adapters*, each of which wraps a specific container. Any container method can be invoked on the unified environment, and depending on the service type is redirected to the container which handles the service. While the server administrator or corresponding tools communicate with the services through UHE, clients still communicate with each container directly, thus the introduced overhead is being minimised.

A possible extension here is to introduce a recursive-hierarchical distributed UHE by adding a UHE adapter which accepts all package formats and redirects requests to other UHE installations, thereby creating a *composite* environment. Due to the many implications like globally unique identifiers, we will not discuss this topic in this paper.

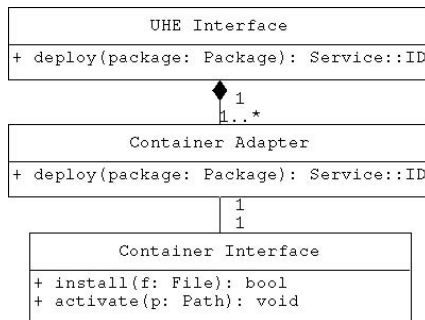


Fig. 4. Design pattern notation of container abstraction

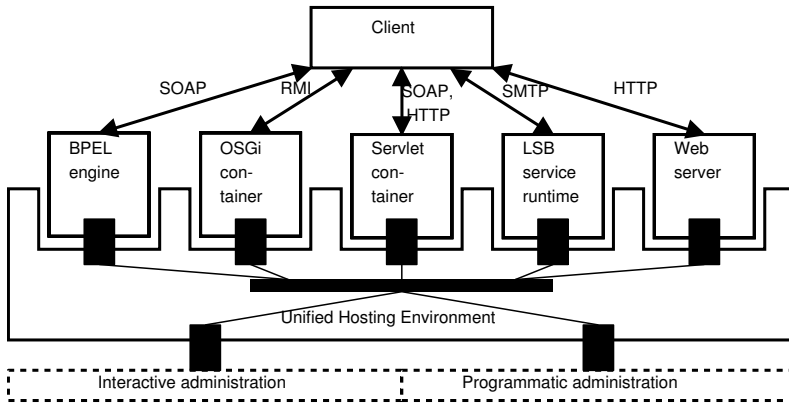


Fig. 5. Hosting environment architecture

In the following subsections, we will explain the implications of the unification on a number of aspects in hosting environments.

6.1 Deployment

Service packages are assumed to be either self-contained, as can often be seen with servlets and associated libraries in WAR packages, or to rely on a dependency resolution and configuration mechanism as is the case with LSB package managers or OSGi bundle loaders [2]. We thus define the *self-contained* property of the package and the *dependency-resolving* property of the respective container. In both cases, additional restrictions have to be assumed about the existence and versions of the installed system software, including language-level virtual machines, container behaviour and system call interfaces. This assumption can be weakened if the implementation supports a recognition of the required environment configuration and setup of a matching environment using a virtualised operating systems or other techniques. This distinction shall be expressed by the *virtualisable* property of the container, and the corresponding *self-described* property of the package.

Therefore, the following property matches are considered a requirement depending on the desired universality of the hosting environment: If any package is not *self-contained*, then the container must be *dependency-resolving*, otherwise the service will not run. If a package is not *self-described*, then the container must meet all its implicit requirements, otherwise the service will not run either.

The table 3 shows the varying degree of self-containedness and self-description of service packages, corresponding to the containers in the previous tables. It is interesting to see that no service package format mandates self-containedness, yet a couple of them usually ship with dependency libraries while others do not.

6.2 Monitoring and Adaptation

A number of software components on a typical hosting system can be considered stakeholders in a unified environment. Adaptation mechanisms, for example, implement abstract adaptation strategies for a controlled change of either the system, the services or

Table 3. Properties of service packages

Service package properties			
	Package type	self-contained self-described	
1.	OSGi bundle	possibly	yes
2.	Axis WAR	possibly	no
3.	Webapp/DEB	no	yes
4.	System/RPM	no	yes
5.	ODE BPEL archive	no	yes
6.	SCA composite	possibly	no
7.	VM image	yes	no

contract offers and running contracts, within well-defined limits. Likewise, monitoring agents need ways to capture system-level, container-level and service-level indicators about running service instances, avoiding to directly provide supporting implementations for the growing variety of containers and preferring to use the available query interfaces. Such requirements reach beyond a unification on the messaging level, as is provided by enterprise service buses. UHE is a suitable layer to unify monitoring and adaptation aspects. Adaptation mechanisms on the middleware level may include load balancing [7]. In such cases, having a unified interface to deploy packages across machines is beneficial as well.

6.3 Testing

Another requirement from hosting companies is that each service must first run in a quarantined environment, also known as sandbox, to find out potential issues with the implementation and to test the acceptance of the service. None of the containers we have evaluated supports a sandbox, although most provide methods to deploy and undeploy service implementations.

Therefore, we consider it mandatory and architecturally sound to provide such a feature within the UHE based on the deployment methods, as shown in Fig. 6. Combined with service versioning, the addition of a testing step supports the long-term

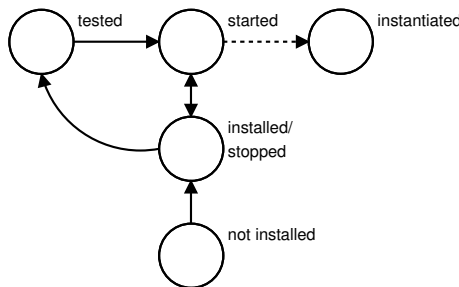


Fig. 6. Service state transitions

management and evolution of service offerings. Migrating an installed and configured service into the production container will take a small amount of work compared to the overall configuration efforts [9].

6.4 Interfaces

A hosting environment should be accessible both by humans and by software. In both cases, the interfaces should not depend on the actual variety and number of containers installed. Instead, the interface should remain stable whenever new technology demands the addition or removal of containers, thus guaranteeing the same level of usability.

For administrators, an interactive interface for basic management tasks like uploading services, listing installed services and running instances and modifying the service states is proposed. It could be implemented as a web interface or as a desktop application integrated into a general system management framework. In addition, we propose having a suitable web service interface to the UHE so that its service management capabilities can be offered to any remote agent. Among the potential users, a service registry could benefit from a formal way of interacting with the containers which collectively can be considered the service repository.

7 Discussion and Conclusions

We have explored commonalities and disparities between various service package formats and service containers acting as hosting environments. Based on a comparison, we presented an abstract view on their structures and abilities, yielding the models TSM for dynamic service packages and TSCM for containers handling the executable artifacts from these packages. Using a formal notation, we were able to define the notion of a Unified Hosting Environment which acts as a proxy and adapter to all concrete service containers. UHE allows for deployment and management of heterogeneous services without any additional service development effort and run-time execution overhead. The concept of UHE is complementary to recent progress towards development methodologies for distributable services. It is supported by specialised implementation techniques with mobile code [6] and the tradable service archive format SAR.

Conceptual extension points including contract handling, service allocation and distributed operation have been identified and will be analysed regarding their unification potential in the future. A prototypical implementation UHE has been created to verify the scope of the models TSM and TSCM in practice¹. The prototype and its further development will help service providers to keep up with the high variety of development methods and packaging formats in service offerings. The easier service hosting becomes, the faster the vision of an *Internet of Services* can be turned into a real infrastructure.

Acknowledgements. The project was funded by means of the German Federal Ministry of Economics and Technology under the promotional reference “01MQ07012”. The authors take the responsibility for the contents.

¹ UHE implementation in the SPACE project: <http://serviceplatform.org/wiki/Puq>

References

1. Bhme, H., Saar, A.: Integration of heterogenous services in the Adaptive Services Grid. In: GI-edn. 2nd International Conference on Grid Service Engineering and Management, Erfurt, Germany. Lecture Notes in Informatics (LNI), NODE 2005/GSEM 2005 p. 69 (September 2005)
2. Cosmo, R.D., Trezentos, P., Zacchiroli, S.: Package upgrades in FOSS distributions: details and challenges. In: First ACM Workshop on Hot Topics in Software Upgrades (HotSWUp), Nashville, Tennessee, USA (October 2008)
3. Harrison, A., Taylor, I.: Dynamic Web Service Deployment Using WSPeer. In: Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies, Baton Rouge, Louisiana, USA, pp. 11–16 (February 2005)
4. Krmer, B.J.: Component meets service: what does the mongrel look like? In: Innovations Syst. Softw. Eng., pp. 385–394. Springer, Heidelberg (November 2008)
5. Lee, S.P., Chan, L.P., Lee, E.W.: Web Services Implementation Methodology for SOA Application. In: Proceedings of the 4th IEEE International Conference on Industrial Informatics (INDIN) Singapore (August 2006)
6. Liu, P., Lewis, M.J.: Mobile Code Enabled Web Services. In: Proceedings of the IEEE International Conference on Web Services (ICWS), Orlando, Florida, USA, pp. 167–174 (July 2005)
7. Lodi, G., Panzieri, F., Rossi, D., Turrini, E.: SLA-Driven Clustering of QoS-Aware Application Servers. *IEEE Transactions on Software Engineering* 33(3) (March 2007)
8. Petrie, C.: The World Wide Wizard of Open Source Services. In: Fourth International Workshop on Semantic Web for Services and Processes (SWSP), Salt Lake City, Utah, USA (July 2007)
9. Sethi, M., Kannan, K., Sachindran, N., Gupta, M.: Rapid Deployment of SOA Solutions via Automated Image Replication and Reconfiguration. In: IEEE International Conference on Services Computing, Honolulu, Hawaii, USA (July 2008)
10. Trger, P., Meyer, H., Melzer, I., Flehmig, M.: Dynamic Provisioning and Monitoring of Stateful Services. In: Proceedings of the 3rd International Conference on Web Information Systems and Technologies - WEBIST, Barcelona, Spain (March 2007)
11. Villanueva, O.A., Touch, J.: Web Service Deployment and Management Using the X-bone. In: Spanish Symposium on Distributed Computing (SEID) Ourense (September 2000)
12. Winkler, M.: Service Description in Business Value Networks. In: Doctoral Symposium of the 4th International Conference Interoperability for Enterprise Software and Applications (I-ESA), Berlin, Germany (March 2008)

Service Differentiation in Multi-tier Application Architectures

Mursalin Habib¹, Yannis Viniotis^{2,3}, Bob Callaway^{2,3}, and Adolfo Rodriguez³

¹ Qualcomm Inc. San Diego, CA 92121, U.S.A.

² Department of Electrical and Computer Engineering, North Carolina State University
Raleigh, NC 27695, U.S.A.

³ IBM, Research Triangle Park, NC 27709, U.S.A.
mhabib@qualcomm.com, candice@ncsu.edu
{rcallawa,adolfo}@us.ibm.com

Abstract. Enterprise networks, such as back-end offices, data centers or web server farms support multiple services and are typically architected in multiple computing tiers. In Service-Oriented-Architecture (SOA) environments, one tier is used for, say, offloading the CPU-intensive XML processing. The business logic is then implemented in a separate tier. The offload tier is typically implemented as a cluster of (potentially virtual) middle-ware appliances. Service differentiation in enterprise networks addresses the issues of managing the enterprise network resources in order to achieve desired performance objectives. In this paper, we define and evaluate via simulations an algorithm that manages allocation of CPU time in the appliance tier, by means of activating and deactivating service domains in the appliance cluster. The main design objective of our algorithm is to overcome the disadvantages of the present static solutions.

Keywords: Middleware, Service oriented architecture, Service differentiation, Multi-tier Application architectures.

1 Introduction

Service oriented architectures (SOA) have become the technology of choice for satisfying many business goals in terms of flexibility, software reuse, and addressing complexity [1], [2]. A way of adopting SOA is through exposing functionality as Web Services. These services leverage the ubiquitous nature of XML as a universal message format; however, this choice often imposes increased computational overhead due to XML parsing. For this reason, enterprise network administrators deploy specialized, hardware-assisted appliances for XML processing. These appliances, called middleware appliances or SOA appliances, are positioned on the edge of the enterprise network, as a separate tier “in front of” the service tier. They are generally deployed in multiples to provide sufficient processing power and to meet high availability requirements. Fig. 1 depicts an abstraction of such an environment that emphasizes the clustering of appliances and servers into two separate computing tiers.

1.1 Service Differentiation

Typically, an enterprise network supports multiple classes of service requests (also known as *service domains*) [10], [3]. For the purposes of this paper, and at a

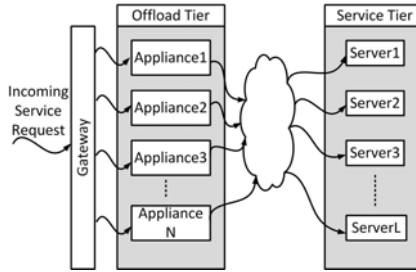


Fig. 1. Abstract architecture of a two-tier enterprise system

high-level, a service domain corresponds to a deployed application or related applications. *Service differentiation* refers to the generic problem of managing the enterprise network resources in order to achieve desired performance objectives on a per domain basis. For example, resources may include the CPU processing power at the appliance tier and/or the service tier; performance may be defined in terms of throughput for one service domain or average delay for another.

It is up to the enterprise network administrator to properly manage (that is, configure and provision) the system resources together as a collective whole, to achieve service domain differentiation. In this paper, we focus on the issue of managing the “island” of middleware appliances. More specifically, we consider the problem of *Unqualified Service Differentiation* that can be stated as follows: “allocate a desired percentage of the CPU power of the appliances to a given service domain”. For example, assuming only three domains SD1, SD2 and SD3, and two appliances with one unit of CPU each, a desired allocation of processing power may be 50%, 30% and 20% respectively.

1.2 Mechanisms for Service Differentiation

A variety of mechanisms can be used to effect this allocation. For example, one such mechanism is “priority-based” CPU scheduling of service domains (see for example, [8]). This mechanism requires per domain buffering and is typically used in the server tier. Another one, used more often in inexpensive appliance devices without built-in intelligence (e.g., FIFO buffering for all domains) and without CPU scheduling, is “*activation/deactivation*” of service domains at the gateway: if more CPU resources are needed to achieve its goal, a service domain can be activated at additional appliances; or, more instances of the domain can be activated at the same appliance. Similarly, a service domain can be deactivated from a subset of the appliances if it exceeds its performance goal. In that sense, activation/deactivation of service domains can be seen as an attempt to *alter the rate* at which requests are allowed to enter the system from the gateway. Allocation of CPU resources is controlled *indirectly*, since it is well-known that a service domain with rate λ and average service time ES will achieve a utilization of $\lambda \cdot ES$ (in a stable system).

To the best of our knowledge, there are two known solution approaches for providing differentiated services via activation/deactivation actions, as we describe in section 2.1. In summary, both known approaches result in (a) inefficient use of appliance resources, and, (b) the inability to provide service differentiation. We address both issues

in this paper. We describe how we could effect *dynamic* provisioning of service domains amongst a cluster of appliances. This way, unlike the known solutions, service domains are not statically bound to a particular (subset of) appliances.

In summary, the main contribution of our research is an algorithm that, unlike the presently known solutions, has the following advantages: (a) it is capable of providing arbitrary allocation of CPU resources to service domains, thus achieving true service differentiation, (b) it utilizes appliance resources in an efficient manner, and thus it leverages processing white-space across all appliances, (c) it increases service locality, and, (d) it does not require manual configurations.

The paper is organized as follows. In Section 2, we provide the system architecture and formulation of the problem. In Section 3, we outline the proposed algorithm. In section 4, we summarize the simulation results and answers to the research questions raised.

2 Problem Formulation

The overall architecture of the system under consideration is depicted in Fig. 1. Service Requests from clients arrive, via a generic transport network, to be processed in the system. A **Gateway** is the first entry point of the system we are going to consider. The gateway distributes requests to the appliances. **Service Domains** represent the grouping of different service requests. **Servers** process service requests belonging to different service domains. The servers are organized in a “service tier”, in the shown architecture. (Middleware) **Appliances** are responsible for pre-processing service requests from different service domains.

The appliances have the capability of buffering requests, in order to accommodate bursty traffic; we assume that they process service requests in a FIFO manner and without preemption¹.

In this paper, we focus on the issue of managing the “island” of middleware appliances. More specifically, we consider the problem of *Unqualified Service Differentiation* that can be stated as follows:

Unqualified Service Differentiation: Provide Service domain m with upto a certain percentage, P_m , of CPU cycles in the appliance cluster.

In typical, commercial SLAs, the percentages may differ, based on whether the system operates under normal or overload conditions. For simplicity, we consider the case of a single condition, hence the name unqualified. We propose an algorithm for achieving Unqualified Service Differentiation in section 3. We present some prior work next.

2.1 Prior Work

As discussed in section 1.2, we consider mechanisms that solve the Unqualified Service Differentiation problem via activation/deactivation actions only. Our motivation for

¹ Even with the presence of a CPU scheduling algorithm inside the appliance, when the number of service domains exceeds the number of buffering classes, service requests within a buffering class are still processed in a FIFO manner.

focusing on such mechanisms comes from two reasons: (a) the fact that existing, commercially available appliances utilize this method, and, (b) in systems with large numbers of service domains, multiplexing multiple domains onto the same buffer forces FIFO scheduling. To the best of our knowledge, there are two known solution approaches; both assume the existence of a gateway device (typically an HTTP router/IP sprayer, HRIS) to distribute service load to the appliance cluster. HRIS is a device without deep-content inspection intelligence that simply routes on, say, a URL, and uses IP spraying to send traffic to appliance replicas.

In the first approach, the administrator groups all appliances in a single group and enables them all to process service requests for any given service [11]. The fronting IP sprayer would forward a service request to each of the appliances, routing the request to the appropriate service port for service-specific processing. This approach suffers from a number of drawbacks. First, in enabling all service domains on every appliance, it is much more difficult to effect differentiated services across service domains competing for the same appliance resources. While an IP sprayer can effectively spread the load (based on policy) amongst the different appliances, it cannot gauge the effect of a specific service request on CPU and thus cannot provide differentiated service amongst the competing service domains. For example, if the administrator wishes to allocate up to 50% of total CPU to a particular service domain, the system as whole can only hope to evenly spread across the appliances, which, under overload conditions, leads to each service domain receiving $1/3$ (33%) of the total CPU. A secondary problem is that it becomes nearly impossible to effect any spatial locality with this solution [11], [15], [4], [6], [7]. In the second approach, the administrator may statically allocate a portion of the appliances to each of the service domains. In this case, each appliance is assigned a specific service domain(s) that it will serve. In this way, service requests for a specific service domain are concentrated on specific appliances, thus achieving spatial locality. Further, the administrator can allocate appliances for service domains proportional to the intended capacity (and to some extent priority) for each individual service, thus achieving some level of differentiated service. However, this approach also has a few drawbacks. First, it is difficult to leverage the white space of appliances serving one service for satisfying requests intended for overloaded appliance and its service. That is, under certain conditions, many of the overall system resources may go under-utilized. Second, the allocation process is manual and cannot adapt to changing request rates and prevailing conditions. This could lead to inefficient resource partitioning and ultimately violate intended differentiated service goals [12], [13], [14], [16], [4], [5].

3 Algorithm Description

SAA/SDA is a closed-loop, feedback-based reactive algorithm. It collects periodic performance measurements from the appliances and uses them to alter the rate of the incoming traffic to meet the differentiation goal. To describe the algorithm we need the definitions provided in subsection 3.1.

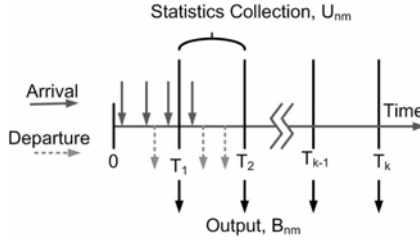


Fig. 2. Decision Instances T_k

3.1 Definitions

The **Provisioning Agent (PA)** is responsible for deciding on activation/deactivation of service domain instances in the appliance cluster. This agent can be implemented as a centralized or distributed application, residing on one or more appliances or a separate compute node. How PA collects the measured statistics from the appliance cluster is out of the scope of this paper.

Decision Instant (T_k) is the k^{th} decision instant at which PA activates/deactivates service domain instances based on the algorithm outcome. As denoted in Fig. 2, at T_k , all the measurements collected in the time interval (T_{k-1}, T_k) are evaluated; activation and deactivation of service domains are enforced. In our simulations, T_k is assumed to form a periodic sequence, for simplicity.

Target CPU % (P_m) is the desired percentage of CPU resources to be allocated to the m^{th} service domain. **Achieved CPU % ($X_m(T_k)$)** is the percentage of the cluster CPU resources obtained by the m^{th} service domain until time T_k .

Down and Up Tolerances DT_m and UT_m : in order to avoid unnecessary oscillations and overhead, when the Achieved CPU % is “close enough” to the Target CPU %, i.e., when

$$P_m - DT_m < X_m(T_k) < P_m + UT_m. \quad (1)$$

the service domain is excluded from activation/deactivation.

Utilization Matrix (U_{nm}) is the achieved resource utilization (e.g., total CPU time used) by the m^{th} service domain in the n^{th} appliance, in the time interval (T_{k-1}, T_k) .

Instantiation Matrix (B_{nm}) is the number of instances of the m^{th} service domain that should be activated in the n^{th} appliance during the time interval (T_{k-1}, T_k) . This is the main decision variable that the PA computes. The mechanism of signalling *HRIS* about the values of B_{nm} and how PA collects the measured statistics from the appliance cluster is out of the scope of this paper.

N is the total **Number of Appliances** in the cluster. M is the **Number of Service Domains** supported by the system.

Groups A and D denote the ranking of service domains. When service domain m is not achieving its Target CPU % (P_m), the PA selects it to be activated in the next decision instant in one or more appliances and thus includes it in Group A . Similarly, when service domain m is allocated more than its Target CPU % (P_m), the PA selects it to be deactivated in the next decision instant in one or more appliances and thus includes it in Group D .

3.2 Algorithm Summary

At each decision instance, at time T_k , $k = 1, 2, \dots$

1. **Collect measurements** (U_{nm}) from the N appliances.
2. **Calculate the actual percentile of allocated resources** for the M service domains using the iterative equation:

$$X_m(T_k) = \frac{1}{kN} \sum_{n=1}^N U_{nm} + \frac{k-1}{k} X_m(T_{k-1}). \quad (2)$$

This equation is a recursive way of calculating the long-term time average of the CPU utilization.

3. **Calculate Thresholding** operations according to Eqn. 1.
4. **Evaluate and Rank Performance** to check if the goal is met. Intuitively, the lower $|X_m(T_k) - P_m|$ is, the “better” the performance of that particular service domain. The service domain is placed in Group A or D as follows. When

$$X_m(T_k) - P_m \geq 0. \quad (3)$$

the service domain meets or exceeds its target and is thus included in Group D .
When

$$X_m(T_k) - P_m < 0. \quad (4)$$

the domain misses its target and is thus included in Group A .

5. **Apply Deactivation Algorithm** to deactivate instances of all service domains in Group D as per algorithm SDA (defined in subsection 3.3).
6. **Apply Activation Algorithm** to activate instances of all service domains in Group A as per algorithm SAA (defined in subsection 3.3).
7. **Feedback** these decisions (expressed as values of the matrix B_{nm}) to the gateway.

The intuition and hope is that, during the next interval (T_k, T_{k+1}), the rate of service requests for a domain m will be favorably affected. Activating “more” instances of a service domain will, hopefully, increase the rate at which requests enter the appliance tier. Thus, the domain will see an increase in its share of the cluster CPU resources; note that the increase may not be obtained during the “next” cycle, due to the effects of FIFO scheduling. Similarly, deactivating instances of a service domain will, hopefully, decrease the rate at which requests enter the appliance tier. Thus, the domain will eventually see a decrease in its share of the cluster CPU resources.

3.3 SAA/SDA Activation and Deactivation Algorithm

There is a myriad of choices in how activation and deactivation of service domains can be enforced. We briefly outline only one choice here; due to the lack of space, we omit specifications of what actions should be taken in certain “special cases”. For more choices and a more detailed description of implementation ramifications, see [9].

1. (SDA) Deactivate one instance of every service domain in Group D in appliances which run service domains in Group A to free up CPU cycles utilized by domains in Group A .²

² One of the omitted special cases specifies that deactivation of a domain should not take place if this action leaves the service domain with zero instances active.

2. (SAA) Using the instantiation matrix B_{nm} , activate one instance of every service domain in Group A , in appliances which run service domains of Group A .

Note that both SDA and SAA will result in a change of the values stored in the matrix B_{nm} . As an example, suppose that we have 4 appliances and 5 service domains with target CPU percentages set at $\{35\%, 25\%, 15\%, 10\%, 5\%\}$. Suppose that the initial value for the instantiation matrix is given by

$$B = \begin{bmatrix} 1 & 0 & 10 & 1 & 4 \\ 10 & 3 & 1 & 2 & 4 \\ 0 & 5 & 8 & 4 & 1 \\ 0 & 0 & 2 & 4 & 10 \end{bmatrix} \quad (5)$$

Suppose that the collected U_{nm} values result in actual CPU percentages $X_m(T_k)$ equal to $\{11\%, 8\%, 19\%, 11\%, 19\%\}$. The tolerances for thresholding are set at 2%, so the algorithm calculates group $A = \{1, 2\}$ and group $B = \{3, 5\}$. Therefore, we must activate domains 1 & 2 and deactivate domains 3 & 5. Now based on the algorithm described (SDA), there is no instances of domains 1 and 2 activated in appliance 4, so there is no need to deactivate instances of domains 3 & 5 in that appliance. However, as there are instances of domains 1 and 2 running in appliances 1, 2 and 3, there will be deactivations of domains 3 and 5 in these appliances. Note that, because there is only one instance of domain 3 activated in appliance 2 and only one instance of domain 5 activated in appliance 3, these two entries will be kept unchanged. Because of the deactivation, as some of the CPU resource utilized by domain 3 and 5 is freed up, under-utilized domain 1 and 2 can take advantage of that and activate one more instance of domain 1 and 2 in appliance 2, domain 1 in appliance 1 (domain 2 cannot be activated in appliance 1 as it is not already activated there) and domain 2 in appliance 3. So, after SDA, we will get (changed values are in bold face),

$$B = \begin{bmatrix} 1 & 0 & \mathbf{9} & 1 & \mathbf{3} \\ 10 & 3 & 1 & 2 & \mathbf{3} \\ 0 & 5 & \mathbf{7} & 4 & 1 \\ 0 & 0 & 2 & 4 & 10 \end{bmatrix} \quad (6)$$

and after SAA, we will get instantiation matrix as follows (changed values are in bold face),

$$B = \begin{bmatrix} \mathbf{2} & 0 & 9 & 1 & 3 \\ \mathbf{11} & \mathbf{4} & 1 & 2 & 3 \\ 0 & \mathbf{6} & 7 & 4 & 1 \\ 0 & 0 & 2 & 4 & 10 \end{bmatrix} \quad (7)$$

4 Simulation and Analysis

4.1 Simulation Goals and Assumptions

Despite the strong engineering intuition, we have no theoretical proof that the SDA/SAA algorithm will be able to satisfy any arbitrary, desired values of CPU allocations.

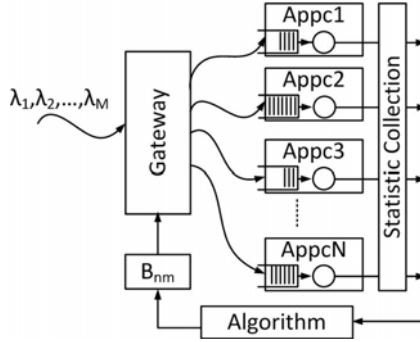


Fig. 3. Simulator Design

Therefore, in order to verify the proposed algorithm, we evaluated the multi-service multi-appliance system by developing a discrete-event simulator in C. We focused our analysis in this paper on the following three sets of questions:

- Q1. Does SDA/SAA “work” (i.e., can it meet the P_m service differentiation goals?) Fig(s). 4 and 5 are representative results in this regard.
- Q2. Is SDA/SAA indeed “better” than the other open-loop, static approaches (i.e., does it have the advantages described in section 2.1)? Fig. 6 (partly) answers this question.
- Q3. How do algorithm parameters (i.e., UT_m/DT_m , N , M , $\{P_m\}$, initial B_{nm} values) affect the behavior of SDA/SAA? Fig(s). 7 through 12 partly answer this question.

The simulation model is depicted in Fig. 3. The service requests arrive at the system in a random fashion. The gateway arrival process for service domain m is modeled for simplicity as a Poisson process³ with arrival rate λ_m . The CPU service time for requests from domain m is a uniform random variable with average value ES_m . For simplicity, all appliances are considered homogeneous. They employ a single, infinite-capacity FIFO buffer for all domains activated in them; their CPU capacity is normalized to 1 unit. Therefore, the CPU utilization of (and thus the CPU allocation to) a service domain m would be $\lambda_m \cdot ES_m$.

4.2 Simulation Results and Analysis

Due to lack of space, in this paper we only include representative results. A more comprehensive set of results and analysis (including confidence intervals) are provided in [9].

To answer question **Q1**, we varied the number of appliances, N from 1 to 10; the number of service domains, M from 1 to 20. For the results depicted in Fig. 4, we set $N = 4$, $M = 3$, the desired goals are $\{P_m\} = \{44\%, 33\%, 22\%\}$ with 2% up and down threshold tolerances. All domains have the same service times and arrival

³ Since the system is controlled in a closed-loop fashion, the nature of the randomness in the arrival (and service time process) is not that critical.

rates. We initialized the instantiation matrix to the same values in all four appliances; in order to create an “unfavorable” situation for the algorithm, the number of instances initialized were $\{2, 5, 10\}$ for the three domains respectively, as opposed to the desired ratios of 44/33/22 respectively. Fig. 4 shows that the SDA/SAA algorithm meets the desired goal despite the unfavorable initial instantiation in the entire cluster. In this simulation, the total arrival rate was chosen high enough to keep the CPUs busy, hence the total utilization (also shown in Fig. 4) approaches 100%.

In Fig. 5, we observe how the algorithm alters the number of instances of service domains in the appliances to achieve the goal. In all figure(s) that depict activated instances, values for appliance 1 are shown (graphs are similar for other appliances). The algorithm causes oscillation in the beginning as for lower value of k , $X_m(k)$ changes abruptly which in turn causes oscillations in the values of B_{nm} .

We demonstrate advantages (c) and (d) mentioned in section 2.1 in detail in [9]. In this paper, to answer question **Q2**, observe that desired P_m goals depend heavily on the actual arrival rates (which may not be known in a real system). For example, suppose we specify $\{P_1, P_2, P_3\} = \{44\%, 33\%, 22\%\}$ and the arrival rates and the average service times for the three service domains are equal. A static allocation, in this case, would allocate CPU times in the ratios 44% : 33% : 22%, wasting 11% for SD1, depriving SD3 of 33-22=11% and leaving a 22% “white space” (unused CPU resource). Fig. 6 shows how SDA/SAA could achieve an equal allocation of CPU resources in this scenario, with a total CPU allocation of 100%, which would eliminate the white space altogether.

To answer question **Q3** involves varying the algorithm parameters $N, M, UT_m/DT_m, P_m$, initial B_{nm} . In all our experiments, the behavior of the algorithm (i.e., the nature of variations in the B_{nm} values) as well as its performance (i.e., the achieved percentages) did not change as we varied the number of appliances N or the number of service domains M . In the interest of saving space, we show in Fig(s). 7 and 8 some results only for the “boundary cases” $N = 1$ and $N = 10$ we tried. The experiments had the same setting as the one used in question **Q1**. As expected, the results agree with those depicted in Fig. 4.

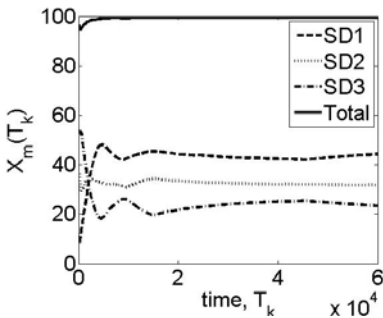


Fig. 4. Utilization $X_m(T_k)$ vs time

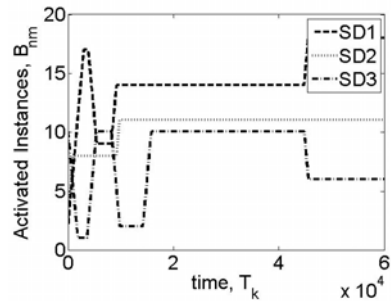


Fig. 5. Variation in B_{nm} values, appliance 1

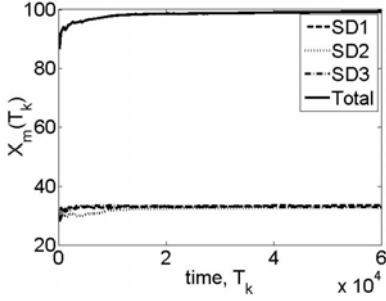


Fig. 6. Utilization $X_m(T_k)$ vs time

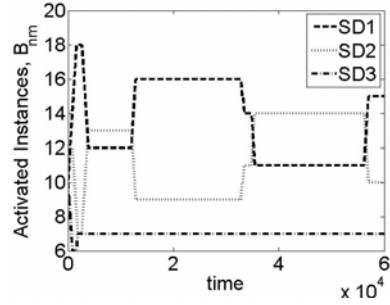


Fig. 7. Variation in B_{nm} Values, in Appliance 1, for $N = 1$ Appliances

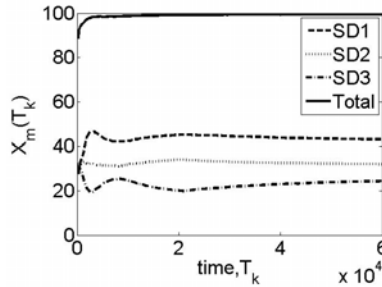


Fig. 8. Utilization $X_m(T_k)$ vs time, in Appliance 1, for $N = 10$ Appliances

The effect of the tolerance parameters UT_m/DT_m is typical of the “oscillatory” behavior depicted in Fig. 10. The figure was produced with (a rather strict) setting of $UT_m = DT_m = 0.1\%$ for all domains; the rest of the experiment setup is the same as the one depicted in the scenario of question Q1. In general, stricter tolerances cause more oscillations in both the goals and the instantiation matrix values (compare Fig. 9 to Fig. 4 and Fig. 10 to Fig. 5). Throughout the experiment, an initial value of $B_{nm} = 10, \forall n, m$ was used.

In general, the P_m parameter can be set by the system administrator in one of two possible ways: “achievable” or “non-achievable”. In the first, the arrival rate λ_m and average service times ES_m of the domain are such that $\lambda_m \cdot ES_m \geq P_m$; in other words, there is enough traffic to take advantage of the allocated CPU resource. Fig. 4 is an example of this case. In the second, we have that $\lambda_m \cdot ES_m < P_m$; in this case, the domain does not have enough traffic to take advantage of the allocated CPU resource. As with all feedback-based algorithms, this situation may “mislead” the algorithm into always activating additional instances of the domain, causing “instability” and eventually affecting other domains too⁴.

Fig. 11 exemplifies what can happen when “non-achievable” goals are set. In this experiment, we set again $\{P_1, P_2, P_3\} = \{44\%, 33\%, 22\%\}$. The arrival rate for SD1

⁴ This is one of the “special cases” we alluded to in section 3.3.

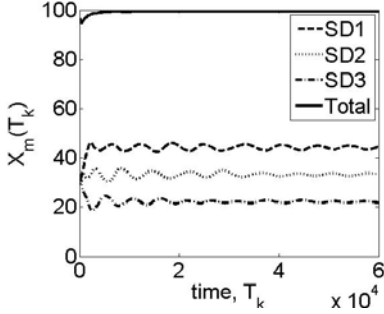


Fig. 9. Utilization $X_m(T_k)$, effect of strict tolerances

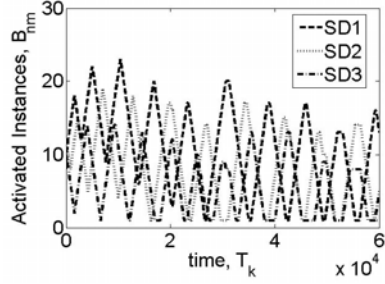


Fig. 10. Variation in B_{nm} , effect of strict tolerances

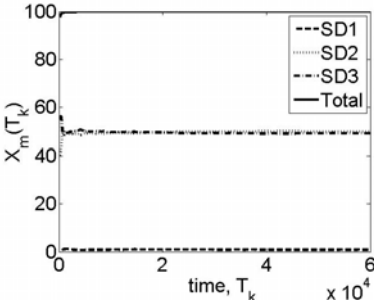


Fig. 11. Utilization $X_m(T_k)$, “non-achievable” P_m goals

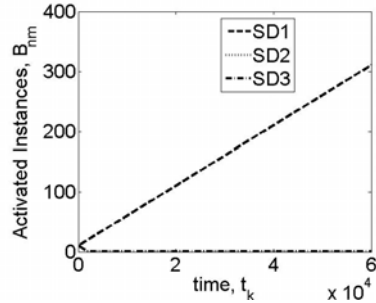


Fig. 12. Potential for instability, “non-achievable” P_m goals

was set low, so that this domain would never reach a 44% CPU utilization, even if it was given full access of the CPUs; its maximum utilization will eventually approach $\lambda_1 \cdot ES_1 \approx 6\%$ in this experiment. The other two domains produced enough traffic to fully utilize their desired percentages. As Fig. 11 shows, these two domains (over)achieve their desired percentages. Figure 12 explains why. The algorithm keeps activating instances for SD1, the “underachieving” domain, at the expense of the other two domains, which are left with only one activated instance each; this explains why these two domains get an equal share of the CPU. The total CPU utilization stays at 100%, as shown in Fig. 11, eliminating any white space.

5 Conclusions

In this paper, we proposed SAA/SDA algorithm, a closed-loop, feedback-based algorithm that provides service differentiation based on CPU utilization measurements in a cluster of middleware appliances. The appliances employ FIFO buffering and thus differentiation is controlled by activation/deactivation of service domains. The algorithm achieves the differentiation goals by controlling the rate at which service requests

are sent to individual appliances in the cluster; it does not rely on a priori knowledge of service domain statistics. It has the following advantages: (a) it is capable of providing arbitrary allocation of CPU resources to service domains, thus achieving true service differentiation, (b) it utilizes appliance resources in an efficient manner, and thus it leverages processing white-space across all appliances, (c) it increases service locality, and, (d) it does not require manual configurations. We have demonstrated such advantages with extensive simulations.

References

1. Erl, T.: *Service Oriented Architecture: A Field Guide to Integraing XML and Webservices*. Prentice Hall, Upper Saddle River (2005)
2. Huhns, M., Singh, M.P.: *Service Oriented Computing: Key Concepts and Principle*. In: *IEEE Internet Computing*, pp. 75–82. IEEE Computer Society, Los Alamitos (2005)
3. Chandrashekar, J., Zhang, Z.-L., Duan, Z., Hou, Y.T.: *Service Oriented Internet*. In: *Proceedings of International Conference on Service Oriented Computing*, pp. 75–82. IEEE Computer Society, Los Alamitos (2003)
4. Kallitsis, M.G., Callaway, R.D., Devetsikiotis, M., Michailidis, G.: *Distributed and Dynamic Resource Allocation for Delay Sensitive Network Services*. In: *Proceedings of 51st Annual IEEE Global Telecommunications Conference (GLOBECOM)*, New Orleans (2008)
5. Callaway, R.D., Devetsikiotis, M., Kan, C.: *Design and Implementation of Measurement-Based Resource Allocation Schemes Within The Realtime Traffic Flow Measurement Architecture*. In: *Proceedings of IEEE International Conference on Communications (ICC)*, Paris (2004)
6. Wang, X., Lan, D., Wang, G., Wang, X., Ye, M., Chen, Y., Wang, Q.: *Appliance-based Autonomic Provisioning Framework for Virtualized Outsourcing Data Center*. In: *Fourth International Conference on Autonomic Computing, ICAC 2007* (2007)
7. Wang, X., Lan, D., Wang, G., Wang, X., Ye, M., Chen, Y., Wang, Q.: *An autonomic provisioning framework for outsourcing data center based on virtual appliances Cluster Computing*, pp. 229–245. Springer Science+Business Media, LLC, Heidelberg (2008)
8. Parekh, A.K., Gallager, R.G.: *A Generalized Processor Sharing Approach to Follower Control in Intergrated Service Networks: The Single Node Case*. *IEEE/ACM Transactions on Networking* 1(3), 344–357 (1993)
9. Habib, M.: *Service appliance provisioning algorithm*, M.S. Thesis, North Carolina State University, Raleigh, NC, USA (2009)
10. Menascée, D.A., Barbará, D., Dodge, R.: *Preserving qos of e-commerce sites through self-tuning: a performance model approach*. In: *Proceedings of the 3rd ACM conference on Electronic Commerce, EC 2001*, New York, NY, USA, pp. 224–234 (2001)
11. Zhu, H., Tang, H., Yang, T.: *Demand-driven service differentiation for cluster-based network servers*. In: *Proc. of IEEE INFOCOM*, pp. 679–688 (2001)
12. Sharma, A., Adarkar, H., Sengupta, S.: *Managing qos through prioritization in 72 web services*. In: *Proceedings on Fourth International Conference on Web Information Systems*, pp. 140–148 (2003)
13. Ranjan, S., Rolia, J., Fu, H., Knightly, E.: *Qos-driven server migration for internet data centers*. In: *Proc. Tenth IEEE International Workshop on Quality of Service*, pp. 3–12 (2002)

14. Zhang, C., Chang, R.N., Perng, C.-S., So, E., Tang, C., Tao, T.: Leveraging service composition relationship to improve cpu demand estimation in soa environments. In: Proceedings of the 2008 IEEE International Conference on Services Computing, SCC 2008, pp. 317–324. IEEE Computer Society, Washington (2008)
15. Wang, X., Du, Z., Chen, Y., Li, S., Lan, D., Wang, G., Chen, Y.: An autonomic provisioning framework for outsourcing data center based on virtual appliances. *Cluster Computing* 11(3), 229–245 (2008)
16. García, D.F., García, J., Entrialgo, J., García, M., Valledor, P., García, R., Campos, A.M.: A qos control mechanism to provide service differentiation and overload protection to internet scalable servers. *IEEE Transactions on Services Computing* 2, 3–16 (2009)

Lessons Learned on the Development of an Enterprise Service Management System Using Model-Driven Engineering

Rodrigo García-Carmona¹, Juan C. Dueñas¹, Félix Cuadrado¹, and José Luis Ruiz²

¹ Universidad Politécnica de Madrid, ETSI Telecomunicación
Ciudad Universitaria s/n. 28040, Madrid, Spain
{rodrigo, jcduenas, fcuadrado}@dit.upm.es

² Indra, c/José Echegaray 8, 28108, Parque Empresarial, Las Rozas, Madrid, Spain
jlrrevuelta@indra.es

Abstract. MDE (Model-Driven Engineering) techniques and tools promise to reduce the complexity and effort of software development. However, although this approach is widely known, there are few reports on its application to real enterprise developments. In this article we present our experience in the creation of an enterprise service management system using MDE. This is a complex system, as it must cope with the heterogeneity and distribution of both software services and their runtime infrastructure. Also, enterprise systems must support multiple non-functional requirements. These requirements are usually fulfilled by enterprise framework solutions, which require a steep learning curve. To overcome these problems we have applied the aforementioned MDE methodologies, starting from a generic information model and partially generating the system from it. We detail the pitfalls found and discuss the strong and weak points of the followed process.

Keywords: MDE, Enterprise Systems, Code Generation, Report on Experience.

1 Introduction

Enterprise service management systems are very complex and costly to develop. Its purpose is the control and automation of the life cycle of software services across a distributed and variable environment. They must adapt to heterogeneous distributed environments, controlling, processing and managing large amounts of data. Also, their internal architecture must support rapid system evolution, in order to keep pace with new business requirements. On top of that, non-functional characteristics such as robustness and security must be maintained.

When we were confronted with the task of developing this kind of system we looked for alternatives in order to simplify its complexity. MDE (Model Driven Engineering) [1] promises to speed the development and reduce complexity by the abstraction of real entities into models, and the application to them of automatic code generation operations. Therefore, we opted to integrate MDE techniques and tools in our development process to try to make use of these capabilities.

In this article we present a report on our experience developing the system. Next section provides an overview over the most important concepts of MDE. Section 3 provides additional information about the target domain, the reasoning behind the adopted approach and the tool selection. The fourth section provides additional details on the case study, detailing the generation processes and system architecture.

Finally, complete discussion on the results and lessons learned after the development is provided, offering some guidelines for similar experiments.

2 Model-Driven Engineering

MDE is a methodology based on the use of abstractions of entities called models. They only contain the information relevant to a particular domain, being oblivious to the remaining details. Their constraints, characteristics and semantics are well defined through metamodels (which are also models), avoiding ambiguities.

The OMG (Object Management Group) is the main standardization organization for MDE languages and processes. Some of its most relevant specifications are MOF [2], a language used for the definition of metamodels, or UML, which is in turn defined using MOF.

MDE processes consist of several kinds of transformations, being model to model and model to text the most prominent. An example model to model transformation allows the enrichment and modification of the definitions of Platform Independent Models (PIM) until they are transformed to Platform Specific Models (PSM). These processes can be automated through the use of transformation languages, such as QVT (Query/View/Transformation).

Code generation activities are the most representative applications of model to text transformations. Under some circumstances, a PSM with enough information can be used to automatically generate the actual source code of the system. In less ideal cases, the generated code base is completed with manual implementation.

Adopting MDE can provide many benefits to the development process. It allows the partial (and in some cases complete) automation of several activities and eases the response to changing requirements or domain specifications. Also, it allows the expression of the problems that need to be solved in a more comprehensible way, providing to architects a clearer view of the system entities.

Applying MDE to the development of enterprise systems has the potential to greatly help in the fulfillment of their particular characteristics [3]. Enterprise management systems present between them many similarities in the software infrastructure and basic requirements, such as communications, or data persistence. These requirements can be captured in model and transformation definitions.

The usage of MDE techniques allows the automation of specific operations and brings “information hiding” principles to the development process, fostering specialization. Work towards solving specific enterprise domain problems using MDE has been performed recently and has shown positive results [4, 5].

However, a considerable effort may be needed for the assimilation of these practices. Thus, the key limiting factor for its enterprise adoption is the availability of a comprehensive and mature tool chain that seamlessly integrates with the development processes and the specific technologies.

3 Case Study Description

3.1 System Requirements

The system under development is an enterprise service management architecture. Its purpose is the control and automation of the life cycle of software products and services across distributed environments. This system will manage information about the physical structure of the target environment, its runtime state, the available software and services, and the dependencies between them. Moreover, it will interact with the physical elements through a well-defined information model, in order to abstract from the complexity and heterogeneity of enterprise systems.

The development of an enterprise system like the one described in this paper is a complex process. The system must be deployed over a distributed environment, and operate with an adequate quality of service, ensuring its high availability, fault tolerance, and scalability. Some representative non-functional requirements are:

- Information consolidation is a fundamental requirement for any management system. Runtime state, statistics, operation logs and system resources must be persisted, sorted and related between each other.
- System components are designed in a decoupled, distributed way, which in turn imposes a need to expose remote communication mechanisms.

As these requirements are common to most enterprise services, in recent years several frameworks and specifications have been developed to provide pre-packed solutions to these aspects. In fact, they have been so useful that its popularity has turned them into additional requirements for the developed services. However, the result is a framework sprawl where the complexity has shifted from the original requirements to a well-established architecture and technology base.

3.2 Technical Approach

After analyzing the characteristics and requirements of the system, we tried to address these concerns by adopting MDE techniques and tools in our development process. We wanted to achieve two main objectives: First, by using the code generation capabilities of MDE tools, we tried to reduce the development effort of the described system, improving productivity. Second, by selecting which parts of the system will be generated, we wanted to abstract as much as possible from the non-functional concerns and the enterprise frameworks, which were not familiar to the development team.

There was also an additional factor supporting the adoption of this approach: the existing information model. As this model is the central element of the management system, it must be comprehensively defined in the analysis stage. This will provide us with an initial input for the selected MDE tool chain. However, it is important to note that it only describes the information and not the system behavior.

In order to apply this approach it is necessary to choose a modeling solution. Although the metamodeling has a huge impact in which solution will be selected (must be powerful, flexible and based upon open and widely adopted standards), the specific requirements of our development process will fundamentally impact the tool support for modeling and code generation. We established the following criteria:

- Comprehensive Java code generation functionality from the available models. The system requirements mandate a Java development, supported by several enterprise frameworks.
- Maturity of the tools. An unfinished or beta solution should be discarded, as tracing errors caused by the code generation are very difficult and costly to detect.
- Out-of-the-box transformations for abstracting from the required frameworks and non-functional concerns (e.g. information persistence through ORM frameworks). Manually defined transformations will not be adopted, as they require the acquisition of a deep understanding in both the transformation language and the underlying framework. Because of that, we will partially adopt an MDE approach.
- Quality of documentation and gentle learning curve. As we will work over the MDE tools, a fundamental factor for its selection is the required effort for applying the technology to our specific problem.

3.3 Tool Selection

After comparing the decision criteria with the available models and tools we chose the following options:

We selected EMF (Eclipse Modeling Framework) [6] ECore as the modeling language for the definition of the information model. EMF is a modeling framework that provides both an implementation of EMOF (Essential MOF) named ECore and a set of supporting tools for defined metamodels, which automatically provide editors for defining model instances, a set of transformations between ECore, XSD and Java, XML-based model persistence and unit test cases. EMF is a very mature and popular project, which has fostered a very active open-source community around the project, providing multiple tools, languages and transformations on top of it.

As our system should support heavy workloads and preserve data integrity, we could not use the base XML serialization provided by EMF, needing relational database support instead. Teneo is an EMF extension that provides a database persistence solution by generating a direct mapping between ECore models and Java ORM (Object Relational Mapping) frameworks, automatically generating the mapping files from the ECore elements. Teneo supports two different types of ORM solutions, Hibernate and JPOX/JDO. We used Hibernate because is the de-facto industry standard (and compatible with the EJB 3.0 specification). It also offers a simplified management interface for the relational operations.

Another system requirement is the ability to distribute the components providing a Web Services remote communication layer on top of the business logic. Web Services is the leading standard for enterprise distributed communications. It promotes contract-based design and loose coupling, through well-defined XML documents for both the contract definition and the information exchange. The contract is expressed through WSDL (Web Services Description Language) files.

The format of the messages in Web Services is specified inside the WSDL descriptor by XSD (XML Schema Definition). Since EMF allows the usage of XSD for the definition of metamodels, we wanted to use these XSDs to create part of the WSDL. For the implementation of Web Services we chose Spring Web Services, a contract-first Web Services framework which was part of our enterprise middleware layer.

The selected tools (EMF, Teneo, Spring-WS) partially address our requirements. They support the definition of both models and metamodels and their transformation to database mappings, WSDL files and Java source code. We chose these solutions discarding more generic transformation model tools because of the previously mentioned requirements (out-of-the-box functionality, abstraction from middleware layers, simplicity and ease of learning).

Figure 1 depicts the relations between these tools and how they work to generate the base artifacts for different aspects of the system (logic, persistence, and communications). In the middle box, EMF automates the generation of both Java classes and XSD files which represent the metamodels obtained from the ECore information model. On the data persistence layer, Teneo automates the generation of database mappings and schemas from the same ECore model that was used in EMF.

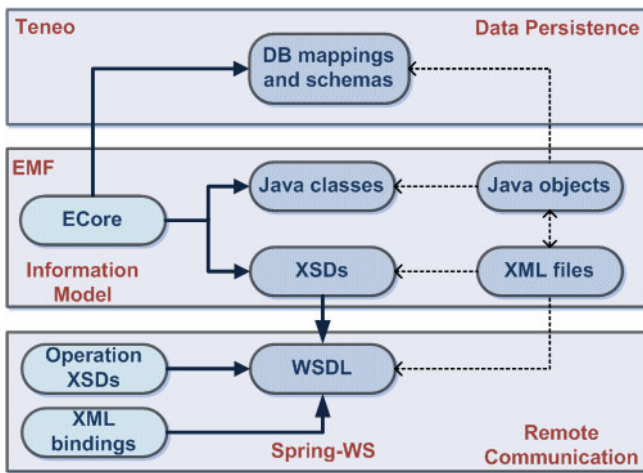


Fig. 1. Transformation flows

Lastly, on the remote communication domain, Spring-WS generates a WSDL descriptor from the XSDs created in the information model layer, XSDs specifying the operations of the interface and XML bindings of the remote interfaces to Java code.

4 Report on Experience

4.1 System Description

As we have described previously, the developed system is a distributed enterprise application, with multiple entities collaborating to provide the required functionality. For its design we have followed a layered architecture, adopting the middleware open-source stack (Spring, OSGi, Hibernate, Web Services) for modular, enterprise applications. The adoption of middleware and framework components greatly reduces the coding effort, and promotes best practices for solving common concerns of every development project.

Figure 2 shows a model-focused structural view of one component of our distributed system. It shows three different areas. The system runs over a runtime environment, formed by hardware, operating system, a Java virtual machine and a set of provided libraries. On top of this substrate reside the models layer. These components are the result of our generation process. Finally, the third group is composed by the actual functionality of the application, the service layer. Developers should focus only on these elements, which are the business logic units, user interfaces, remote services, and inventory services. As the model layer provides automatic transformations it abstracts from the middleware infrastructure in charge of the remote serialization and persistence.

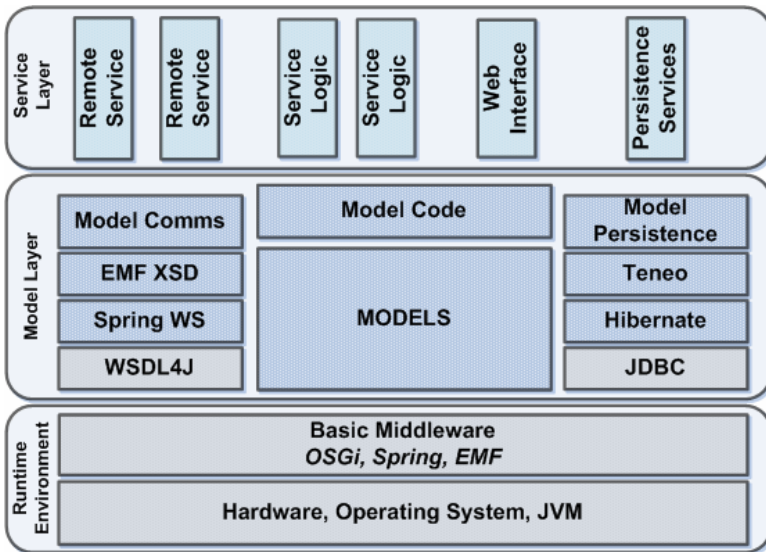


Fig. 2. System structural view

4.2 Process Practices

With the characteristics of the selected tools and the requirements of the system in mind we defined a flow for the detailed design and implementation activities. Figure 3 shows its steps and the transitions between them.

The application of MDE translates into the following tasks:

- Definition of models using MDE models and metamodels.
- Modification of already created models, in order to adapt them; either by using transformations (model-to-model) or by hand (model tuning).
- Generation of code from the models, mainly using model to code transformations.
- Modification of generated code (code tuning).
- Implementation of code not covered by MDE, which in our case pertains to the system logic.
- Testing of both the generated and manually created elements.

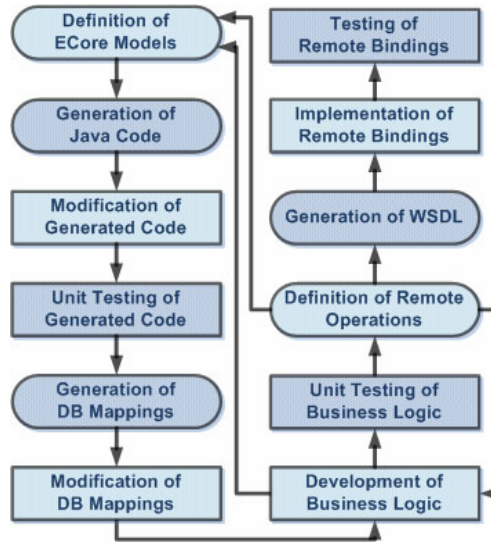


Fig. 3. Development process

Concerning testing it is important to note that EMF generates unit tests that validate the generated source code. Therefore, the testing tasks can be performed with automatically created or hand-written tests.

5 Discussion

This section provides additional discussion on the followed approach after its completion. We will present both a small quantitative analysis of the finished system and a summary of the lessons we have learned. We think this information can be useful to not only evaluate the success of the approach but also improve similar processes.

5.1 Quantitative Analysis

To evaluate the generated code some metrics have been performed. The results of this analysis are depicted in Table 1.

The first two rows contain the most basic information that can be obtained: the raw number of lines of code and Java classes. It is important to note that the size of the modeled part weights roughly half of the system (60.000 lines of code excluding libraries). Most of this code contains the information model and the XML serialization engine.

The remaining rows comprise some software metrics that try to measure the quality of the code. Efferent couplings indicates how focused are the classes. The remaining metrics (cyclomatic complexity, number of lines per method and number of locals) indicate the complexity and comprehensibility of the code. All the values are averages for all the classes or methods.

Since generated and manually written code cannot be compared side by side, we compared the amounts of code of the model definitions and the generated elements. The model definitions span 1609 lines, the ratio is of 20.6 Java lines generated per line of model definition written.

Table 1. Code metrics

Metric	Value
Lines of Code	33125
Classes	290
Average Efferent Couplings	6.91
Average Cyclomatic Complexity	2.06
Average Number of Lines per Method	13.63
Average Number of Locals	1.44

5.2 Lessons Learned

During the process we identified some critical risks for the success of the development with this approach. Most of these pitfalls could be avoided taking some factors into consideration. Further on, we expose the most remarkable issues:

Application of Mature Transformations. Our intent with the described generation process was to take models as a foundation, trying to abstract whenever possible of the specific middleware for the previously described concerns, such as persistence or remote communications.

Although our experience was positive (used these capabilities seamlessly over the model layer), we found some problems using one of the transformation frameworks (Teneo 0.8): its data persistence service did not work as expected in common situations (updating operations). Detection of such failures was difficult because the source of problems could be in any of the layers, and we had to look into their source code, losing the theoretical advantages of abstraction. Therefore, tool and framework maturity are a fundamental risk to be assessed for adopting this type of approach.

Limits in the Abstractions. We were also affected by the law of leaky abstractions [7], as the transformations hid useful concepts in the lower levels that could only be obtained by respecting these low-level constraints in the business logic (lazy loading from the database improves efficiency but imposes session management in the upper layer).

Model Definition Accuracy. The success of the complete development is heavily dependent on this. During our development, an error in the business logic was finally traced to a mistake in the definition of the information model. We expressed a relationship between elements as a composition instead of an aggregation, and the generated code did behave as we defined (but not intended).

Application of Corrective Changes. Probably the most important model transformation that a solution can offer is the generation of code. In our experience almost all the chosen solutions behaved perfectly on this matter. However, the generation process can in some cases be far from perfect and the generated code could not be used directly.

We experienced this drawback with Teneo. The generated mapping files had to be manually edited to solve various problems. The greatest time sink here was to trace the failure to the generated model and figure what tweaks were needed.

On the other hand, the automatic generation of unit test cases that EMF provided helped greatly to discard those models as the source of any failure.

Application of Perfective Changes. Sometimes the generated elements do not accomplish all the goals that have been set. In these situations the missing features have to be implemented into the generated code by hand. If the generated artifacts are well documented and easily readable, applying these improvements is a good way to build over the base functionality.

In our case, the code produced by EMF lacked proper methods for asserting the equality between two elements, managing collections or generating a unique identifier. But fortunately the generated code did not require a deep knowledge of EMF. With the help of annotations to preserve these non-generated methods in future transformations and thanks to the cleanliness and organization of the code, the application of these perfective changes was straightforward.

Cost of Starting a New Iteration. It is very common during the development process to go back to a previous step, introducing some changes and continue from there. This usually forces the redefinition of models and regeneration of code. In these cases it is very important to keep track of all the manual changes and procedures that have to be applied after finishing the automated tasks. For instance: performing the correct code modifications after its regeneration.

Therefore, it is vital to have a detailed and documented process for the development with MDE. We addressed this point by adopting the detailed flow shown in previous sections.

Coverage of Transformations. MDE is based upon transformations. However, special attention needs to be devoted to the system components where the necessary transformations are not automatically performed. These sections must be reviewed after each code regeneration operation and some parts need to be manually implemented.

During the development of the system we found that Spring Web Services, although generated the WSDL, lacked the tools to do the same with the bindings between the logic and the interfaces. In the end we implemented those bindings manually. However, in retrospective we think that defining and implementing these transformations could have been a better solution. The workload would have been similar but in further iterations the benefits of extending MDE coverage would have been considerable.

6 Conclusions

In this case study we have developed a real-world enterprise management system in a model-centric view through MDE processes. This approach has allowed us to implement some non-functional requirements such as remote communications or information persistence with model transformation techniques and tools, using available open source tools and libraries.

The results obtained during this development have been satisfactory. The reduced effort obtained by the code generation capabilities greatly helped to speed the process. The general perception of both the developers and project managers are that the use of these methodologies, albeit the problems faced, has eased the development process and improved the quality of the produced system. It seems clear that the characteristics of the enterprise domain make it perfectly-suited for automating the generation of parts of the system.

However, regarding the level of achieved abstraction from the middleware layers we identified several key factors that greatly impact the results in this area. We believe that our lessons learned in this case study can help with the execution of similar processes and greatly reduce the risks involved and shorten the development cycles.

Acknowledgements. The work presented here has been performed in the context of the CENIT-ITECBAN project, under a grant from the Ministerio de Industria, Comercio y Turismo de España.

References

1. Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. *Computer* 39(2), 25–31 (2006)
2. Object Management Group, Meta Object Facility Specification 2.0.(January 2006), <http://www.omg.org/spec/MOF/2.0/>
3. Frankel, D.S.: *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, Chichester (2003)
4. Quartel, D., Pokraev, S., Pessoa, R.M., van Sinderen, M.: Model-Driven Development of a Mediation Service. In: 12th International IEEE Enterprise Distributed Object Computing Conference, Munchen (2008)
5. White, J., Schmidt, D.C., Czarnecki, K., Wienands, C., Lenz, G.: Automated Model-Based Configuration of Enterprise Java Applications. In: 11th International IEEE Enterprise Distributed Object Computing Conference, Annapolis (2007)
6. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: *EMF: Eclipse Modeling Framework*, 2nd edn. Addison-Wesley Professional, Reading (2008)
7. Spolsky, J.: *Joel on Software*. Apress (2007)

Part II

Software Engineering

Checking Regulatory Compliance of Business Processes and Information Systems

Motoshi Saeki¹, Haruhiko Kaiya², and Satoshi Hattori¹

¹ Dept. of Computer Science, Tokyo Institute of Technology
Ookayama 2-12-1-W8-83, Meguro-ku, Tokyo 152-8552, Japan

² Dept. of Computer Science, Shinshu University
Wakasato 4-17-1, Nagano 380-8553, Japan

{saeki,satoshi}@se.cs.titech.ac.jp, kaiya@cs.shinshu-u.ac.jp

Abstract. In these years, many laws and regulations are being enacted to prevent business processes (BPs) and information systems (ISs) from their malicious users. As a result, it is significant for organizations to ensure that their BPs and ISs comply with these regulations. This paper proposes a technique to apply a formal technique to ensure the regulatory compliance of BP or IS descriptions written in use case models. We translate the use case models of the behavior of BPs and ISs into finite state transition machines. Regulations are represented with computational tree logic (CTL) and their satisfiability are automatically verified using a model checker SMV. The modality of regulations can be specified with temporal operators based on branching time semantics of the CTL in our technique.

1 Introduction

To develop an information system used in an organization, we firstly model a business process so that it can solve the problems that the organization has. We have several techniques to model business processes and information systems, e.g. work flow, state transition, use case and goal modeling etc. In particular, we often specify the behavior of business processes and information systems.

In these years, many laws and regulations (simply, regulations) are being enacted to prevent business processes and information systems from their malicious users. As a result, we should develop a business process and an information system that are compliant with these regulations. If we developed a business process or an information system that was not compliant with the regulations, we could be punished and its compensation could be claimed to us, as a result we could take much financial and social damage. It is significant for us to ensure that our business processes and information systems comply with these regulations. Furthermore, if we would find that the information system that is being developed was not compliant with its related regulations, we have to redo its development and its development cost and efforts seriously increase. Thus we have to check if a behavioral specification of the business process and/or the information system to be developed is compliant with regulations as early as possible, in order to reduce its development cost. In an earlier stage of development, we should verify that the behavioral specification comply with regulations.

We propose the technique to check formally regulatory compliance of the behavioral specifications of business processes and information systems, using a model checker SMV. In [14], in order to detect the possibilities of non-compliance, its authors have developed the technique to represent specification statements and regulatory ones with case frames of Fillmore's case grammar and then to match these case frames. However, it dealt with itemized assertive sentences as specification statements only and it did not consider behavioral aspects such as execution order of actions in the system. In this paper, we model the behavior of a business process or an information system with a finite state transition machine that can be an input to a model checker. Regulatory statements are formally represented with temporal logical formulas and the model checker verifies if these logical formulas are true in the state transition machine or not. If the logical formulas are true, we can judge the behavioral specification to be compliant with the regulations. If the formulas are false, since the model checker outputs counterexamples showing the state transition sequences unsatisfying the regulations, we can recognize where its regulatory non-compliance exists.

The rest of the paper is organized as follows. Section 2 presents how to represent regulatory statements with branching time temporal logic (another name, CTL: computational tree logic, and we use the abbreviation CTL below). In particular, we emphasize the modality of regulatory statements such as obligation, prohibition, etc. and discuss how to formally represent them with the temporal operators of CTL. We also explain the overview of our checking process for regulatory compliance and illustrate its details together with supporting tools in section 3. It includes a description language of state transition machines for the model checker SMV and the terminology matching to retrieve the relevant regulatory statements to be verified. In section 4, we discuss another example to show the usefulness of our approach. Sections 5 and 6 are for related work and concluding remarks respectively.

2 Regulation

2.1 Representing Regulations

A typical example of regulations related to IT technology is Japanese Act on the Protection of Personal Information [1] that specifies the proper handling of personal information such as names, addresses and telephone numbers of persons in order to prevent from making misuse of this information. For example, the Article 18, No. 1 of Act on the Protection of Personal Information provides that

Article 18, No. 1 of Act on the Protection of Personal Information:

When having acquired personal information, an entity handling personal information must, except in cases in which the Purpose of Use has already been publicly announced, promptly notify the person of the Purpose of Use or publicly announce the Purpose of Use.

According to [5], a regulatory statement consists of 1) the descriptions of a situation where the statement should be applied and 2) the descriptions of obligation, prohibition, permission and exemption of an entity's acts under the specified situation. In the above

example, we can consider that “when having acquired personal information, except in cases in which the Purpose of Use has already been publicly announced” is a situation where this act should be applied, while “notify” and “announce” represent the acts of “the entity”. These acts are obligations that the entity should perform.

The first thing that we should address is how to deal with four modalities, obligation, prohibition, permission and exemption using mathematical notation such as formal logic. We use the temporal operators of CTL to represent these modalities. Suppose that we specify the behavior of an information system with a finite state transition machine. Since state transitions occur non-deterministically in it, there exist several execution paths in the information system. When we define the states as nodes and the transitions as edges, we can get a tree called *computational tree* that specifies these execution paths. The properties that hold on the tree can be defined with CTL formulas. Suppose that R is a logical formula. We use four types of temporal operators **AF**, **AG**, **EF** and **EG** and their intuitive meanings are as follows. **AF** R is true *iff* R is eventually true for every path, **AG** R is true *iff* R is always true for every path, **EF** R is true *iff* there is a path where R is eventually true, and **EG** R is true *iff* there is a path where R is always true.

The value of a proposition is either true or false at a node. Let P and Q be propositions of a situation and an act respectively. Q is true if the act is being executed. By using the above four operators, we can represent a regulatory statement with the modalities as follows.

Obligation : $P \rightarrow \mathbf{AF} Q$

Prohibition : $P \rightarrow \mathbf{AG} \neg Q$

Permission : $P \rightarrow \mathbf{EF} Q$

Exemption : $P \rightarrow \mathbf{EG} \neg Q$

The three computational tree of Figure 1 shows why the above formulas can express a modality of a regulatory statement. Gray-colored nodes on the paths stand for the states where Q is true, while each root node denotes the state where the situation P is true. In the case of obligation, we should perform Q if the situation P is true, whatever execution path we take. Therefore Q should be eventually true for every path outgoing from the node P . On the other hand, a regulatory statement of prohibition says that we are not allowed to execute Q on any path. $\neg Q$ should continuously be true on any node of every path outgoing from P , i.e. Q is always false for every path. If there exists a path where Q is eventually true, Q is permitted to be executed. If there exists a path where Q is always false, we are exempted from executing Q . Note that CTL has binary temporal operators based on **until** and we can represent with these operators time intervals such as the deadline when an obligation keeps on holding. For simplicity, we will not refer to them throughout this paper but we can deal with them in the same way.

In the cases of permission and exemption, although the regulatory statement is not true on an information system, we cannot say that it violates the regulation. For example, if “ $P \rightarrow \mathbf{EF} Q$ ” (permission of Q) is not true, there are no paths where Q can be executed. Even though the act Q is *permitted*, we don’t always need to execute Q and non-execution of Q is not a regulatory violation. However, if an information system will not have the function to execute the permitted act Q , it may have a disadvantage to competitors’ products having this function in the market. Moreover, there is a possibility that its users may accuse it of inconvenience because they cannot use the function.

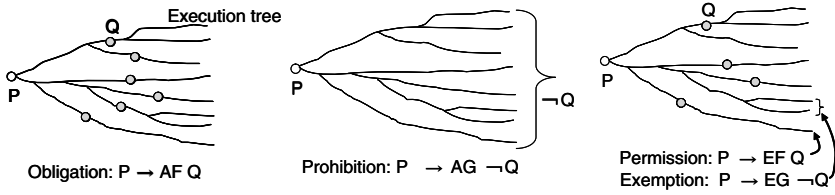


Fig. 1. Computation Tree and Modality of Regulations

In addition, suppose the case when the act Q is executed at every path, i.e. the same situation as obligation. This case is compliant with the regulatory statement. However, if the execution of the act Q brings about inconveniences to the users of the information system, the system where Q is executed in any case may have a disadvantage. Thus we additionally check the execution of Q in every path, using the formula “ $P \rightarrow \mathbf{AG} Q$ ”. Similarly in the case of the exemption of Q , we additionally check whether Q is not executed in any path, i.e. is prohibited or not.

We continue to discuss how to represent a regulatory statement with a CTL formula, using as an example the Article 18, No. 1 of Act on the Protection of Personal Information, mentioned in the beginning of this sub section. This article claims the obligation of the acts “announce” or “notify”. The situation part and the act one in a regulatory statement can be described with logical combinations of case frames as shown in [14]. The technique of case frames was originated from Fillmore’s Case Grammar to represent the semantics of natural language sentences. A case frame consists of a verb and semantic roles of the words that frequently co-occur with the verb. These semantic roles are specific to a verb and are called *case*. For example, the case frame of the verb “get”, having the cases “actor”, “object” and “source”, can be described as “get(actor, object, source)”, where “get” denotes the acquisition of the thing specified by the object case. The actor case represents the entity that performs the action of “get” and that will own the thing as the result of the “get” action. The source case denotes the entity from which the actor acquires the object. By filling these case slots with the words actually appearing in a sentence, we can obtain its semantic representation. In the example of the sentence “an entity handling personal information acquires from a member her personal information”, we can use the case frame of “get” and have “get(entity handling personal information, personal information, member)” as its intermediate semantic representation. Finally, we can represent the example statement of Article 18, No.1 using case frames and CTL as follows;

```

get(x, Personal_information, y)
  ∧ ¬ announce(x, Purpose_of_use)
  ∧ aggregation(y, Personal_information)
  ∧ handle(x, Personal_information, Purpose_of_use)
→ AF (notify(x, Purpose_of_use, y)
      ∨ announce(x, Purpose_of_use))
    
```

Note that the identifiers of lower case characters such as “ x ” and “ y ” stand for variables, and we can fill them with any words. In this sense, the formula can be considered as a template.

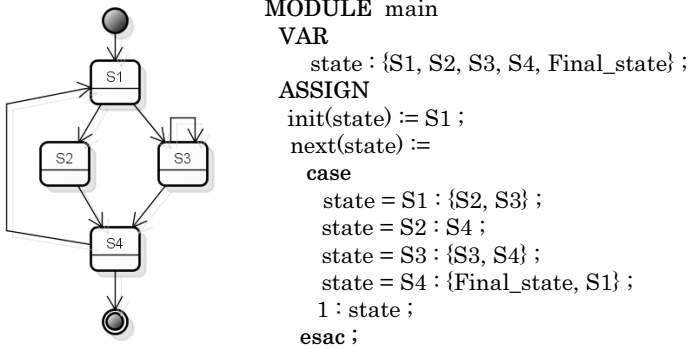


Fig. 2. Description of a FSM with SMV

2.2 Describing FSMs

We use the model checker SMV [9], because it can deal with CTLs. In SMV, a system to be checked is represented as a set of concurrent sequential processes and each process is defined as a non-deterministic finite state transition machine (FSM). In a FSM, state transitions are defined as changes of values of explicitly declared state variables. Figure 2 includes an example of a FSM and its representation with the language of the model checker SMV. The left part of the figure shows a diagram of the FSM, and rounded rectangles and arrows represent states and transitions between them respectively.

The right of the figure shows the description of the FSM for SMV, and it has only one global variable named “state” that stores the current state. The VAR section declares the variable and its domain, i.e. a set of values that can be assigned to the variable. The expression “next(state)” denotes the value of “state” at the next state. A case block (case ... esac) sets the value of the next state according to the current value. The values before and after the colon (:) expresses the current value and the next one respectively. For example, if the current value is S1, the next value is set to S2 or S3 after a transition. The bottom line of the case block includes “1” before the colon and it means “otherwise”.

To make it easy to find the relevant regulatory statements and to generate their CTL formulas from the FSM, it is preferable to use as the FSM’s state names natural-language sentences or phases like use case descriptions, or their case frame notation as shown in the right part of Figure 5.

3 Checking Process

3.1 Overview

Figure 3 shows the process of checking regulatory compliance in a business process or an information system. Its behavior is modeled with a state transition model and the model is described with the language of the model checker. In our approach, we also translate regulatory statements into CTLs as shown in section 2.1, and verify if the CTLs are true on the state transition machine by using a model checker. If the model

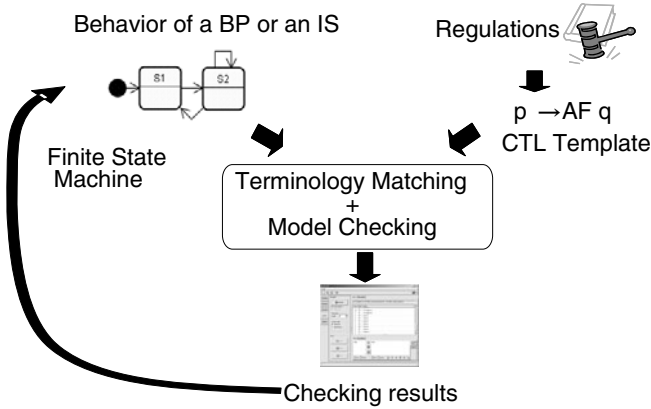


Fig. 3. Overview of a Checking Process

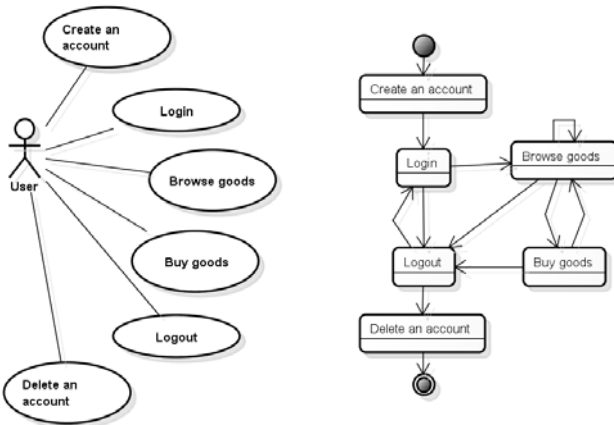


Fig. 4. On-line E-Shop

checker finds that the CTLs are false, it produces the examples where the CTLs are false, i.e. the counterexamples. We can explore the counterexamples and identify which parts of the use case descriptions may cause the detected noncompliance. Note that in the case of permission or exemption regulatory statements, as mentioned in section 2.1, 1) the resulting falseness does not lead to noncompliance and 2) we can perform an additional check even in the case of true.

The words, terms and phrases (terms, hereafter) appearing in regulatory statements are different from the terms in the state transition machines, but they may have the same meaning. We need a task for identify the terms having the same meaning and unify them into a single expression. “Terminology matching” is for matching the terms appearing in the CTLs to those in the state transition machines by using synonym dictionaries such as WordNet. The supporting technique for this task will be illustrated in section 3.3.

Create an account

Actor: User

Pre condition

There are no accounts yet.

Normal flow

1. The user sends his personal information to the system.
2. The system checks the validity of the personal information.
3. The system issues login ID and password to the user.

Post condition

The user gets an account and the user is not logged in yet.

MODULE create_an_account(state)

ASSIGN

next(state) :=

case

state = "there_are_no_accounts_yet" :

"send(User,Personal_information,System)" ;

state = "send(User,Personal_information,System)" :

"check(System,Validity_of_Personal_information)" ;

state = "check(System,Validity_of_Personal_information)" :

"issue(System, Login_ID_and_password,User)" ;

1 : state ;

esac;

(a) Use case description

(b) SMV description

Fig. 5. Detailed Description of Create an account

The image shows a software interface window with a title bar containing standard window controls. The window is divided into several sections:

- state:** Contains the text `send(User, Personal_information, System)`.
- verb:** Contains two buttons labeled `send` and `get`.
- case frame:** Contains a table with columns 'actor', 'object', and 'source'. The first row has values 'get', 'system', and 'personal in...'. Below the table is a button labeled `updateGuide`.
- guide:** Contains the text `Acts to be considered` followed by `notify(system, Purpose_of_use, user)` and `announce(system, Purpose_of_use)`.
- Situations to be considered:** Contains the text `aggregation(user, personal information)` and `handle(system, personal information, Purpose_of_use)`.
- regulation:** Contains the text 'Article 18 (Notice of the Purpose of Use at the Time of Acquisition, etc.)' followed by a numbered list item: '1. When having acquired personal information, an entity handling personal information must, except in cases in which the Purpose of Use has already been publicly announced, promptly notify the person of the Purpose of Use or publicly announce the Purpose of Use.'
- situation:** Contains logical expressions: `get(x, Personal_information, y)`, `∧ aggregation(y, Personal_information)`, `∧ handle(x, Personal_information, Purpose_of_use)`, and `∧ ¬(announce(x, Purpose_of_use))`.
- obligation:** Contains logical expressions: `notify(x, Purpose_of_use, y)` and `∀ announce(x, Purpose_of_use)`.

Fig. 6. Supporting a Terminology Matching Task

3.2 Example

In this subsection, we explain an example that will be used throughout this section. The example is an on-line shop for goods like Amazon, and Figure 4 depicts its use case diagram and its state transition diagram. The overall behavior of the business process of a shop user is specified with the state transition diagram shown in the right part of

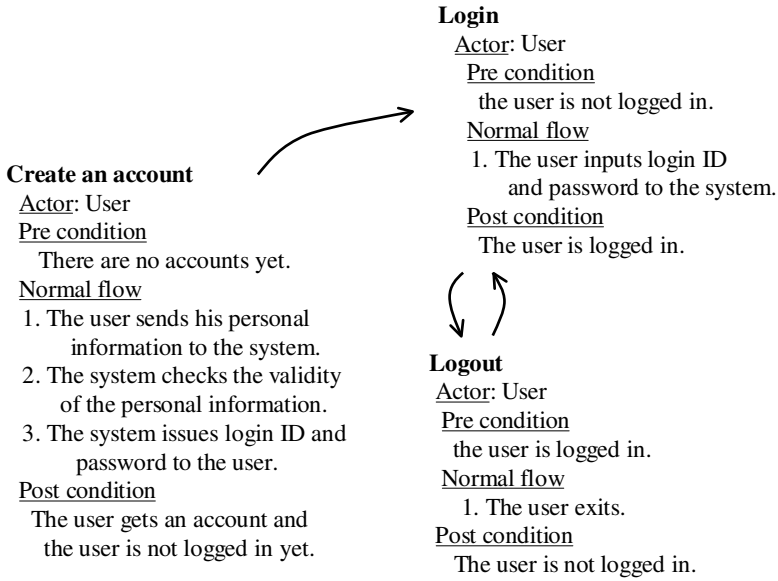


Fig. 7. A Counterexample

the figure. A user first creates her account (Create an account) and then logs in her account (Login). She browses various catalogs of goods to find the items that she want to get(Browse goods). If she finds a good that she like to buy, she buys it by specifying a payment method, e.g. inputting her credit card information (Buy goods). She can browse the catalogs of goods without buying anything (the transition from Browse goods to Logout) and can buy several goods repeatedly in one session (the transition loop between Browse goods and Buy goods). Figure 5 shows the detailed behavior of “Create an account”, which is described as a use case description in the left part and with SMV language in the right part of the figure. Note that, for simplicity, we use the descriptions that are not syntactically allowed in SMV. As mentioned in section 2.2, we consider an action currently executed in the use case as a current state and the global variable “state” holds the name of the currently executed action. If this action finishes and the next action starts being executed, the name of the next action is assigned to the variable. As for pre and post conditions in a use case, by assigning the name of the condition to the variable, we represent the state where it comes to be true. We don’t intend to propose a technique to translate use cases into a state transition machine. In fact, more elaborated translation techniques can be found in [8,15,16] and they may be used.

3.3 Terminology Matching

The goal of terminology matching task is 1) retrieving the regulatory statements relevant to a FSM by unifying terms of the regulatory statements to the terms appearing in the FSM such as state names, and 2) generating the CTL formulas in SMV-acceptable form from the unified formulas of the regulatory statements.

Suppose that a FSM has the state name “send(User, Personal_information, System)”, which is case frame notation of the sentence “The user sends his personal information to the system”. When we use this sentence, we can get a case frame as its semantic representation after lexical analysis. If we use as a state name an incomplete natural language sentence, e.g. “send personal information” where a subject is missing, we should add missing words so as to get its case frame. In this example, the verb “get” in the case frame of Article 18, No.1 is semantically the same as “send” but the flow of the object (personal information) of this act is reverse to “send”. We have a dictionary of case frames, and it includes information on synonym verbs and their case slots. It also has the rules of replacing a verb and its case slot values, keeping the same meaning. For example, a rule says that the frame “get(actor:x, object:y, source:z)” can be replaced with “send(actor:z, object:y, target:z)”. After this replacement, we fill the variables “x” and “y” with “System” and “User” respectively so as to match with the example sentence the resulting case frame (the situation part of the Article 18, No.1). From the context of the FSM, since it is obvious that the system (“x”) handles with personal information and that the user (“y”) has personal information, we can omit the predicates “handle” and “aggregate”. Finally we have the following CTL as the result of this Terminology Matching task.

```
state = "send(User, Personal_information, System)"
→ AF (state = "notify(System, Purpose_of_use, User)"
    ∨ state = "announce(System, Purpose_of_use)")
```

The above is just the CTL formula to be checked if the FSM has regulatory noncompliance or not, and an input to a model checker.

Since Terminology Matching task deals with the semantics of natural language, we cannot fully automate it. However, we have a computerized tool to support this task, based on the technique in [14]. The tool has the following functions; 1) analyzing natural language descriptions and extracting their case structures if necessary, 2) having a dictionary of case frames of regulatory statements, 3) retrieving the regulatory statements which can be matched to the case frames of the state names of the FSM, and 4) generating the CTL formula of regulatory statements by unifying the terms and simplifying it. Figure 6 shows a screen shot of our tool. The tool suggests a pair of the description “send(User, Personal_information, System)” and the related regulatory statements after matching them in case frame level. This matching process uses Japanese-English translation dictionary and synonym ones. We have extracted case frames from the Articles 15 - 36 of Act on the Protection of Personal Information and stored them in a dictionary beforehand. The tool checks if the verb phrases appearing in the descriptions can be matched to these case frames. As a result, the left and right areas of the window of Figure 6 show the description “send(User, Personal_information, System)” of “Create an account” in Figure 5 and the Article 18 No.1 respectively, because they can be matched. Some information helpful to produce the input CTL formula is displayed in the other area of the window, e.g. situations and acts to be considered during producing the formula.

3.4 Model Checking

The model checker of SMV is called NuSMV. Since NuSMV checks if a formula is true at the initial state of a FSM, we attach the operator AG in the head of the formula that was obtained in the Term Matching task.

The NuSMV shows that the CTL of Article 18 No.1 is false, and we can recognize that our example has a regulatory violation because the verified CTL is an obligation. The counterexample that the NuSMV outputs suggests the instance of execution paths where the CTL comes to be false. According to it, as shown in Figure 7 which is described in use case description form not in state transition style for simplicity, we can recognize that the following scenario caused the regulatory violation. After executing Create an account, Login is executed and then its login is successful. After that, the Logout use case is immediately executed. Login and logout are iterated as a loop. This path does not satisfy the CTL of Article 18 No.1 specifying the obligation of the action “notify” or “announce”.

Figure 8 illustrates the verification example mentioned above. Note that we used abbreviated literal forms instead of string data types to represent the values of “state”

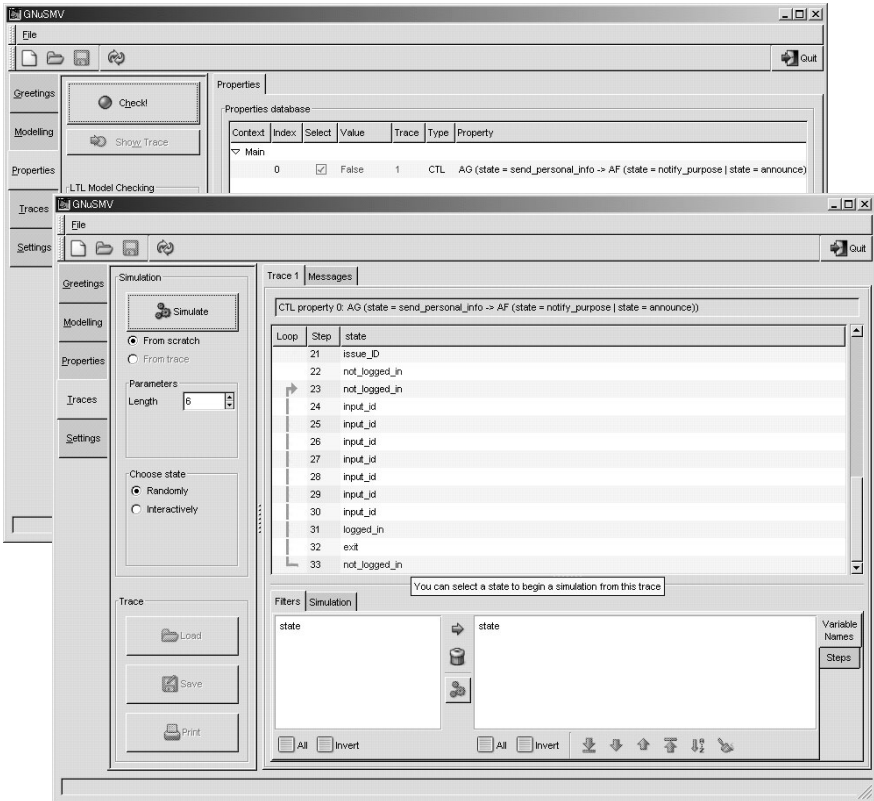


Fig. 8. Verification Example with NuSMV

variable for brevity and the ability of NuSMV. For example, we used the literals `send_personal_info` and `notify_purpose` in the NuSMV tool instead of the strings “`send(User, Personal_information, System)`” and “`notify(System, Purpose_of_use, User)`” respectively. The figure includes two windows; one is the verification result and another is a counterexample. The rear window shows that the CTL of Article 18 No.1 is false (the CTL numbered with 0 in the Property tab), and we can recognize that our example has a regulatory violation. The counterexample is shown in the front window of the figure. According to it, after executing Create an account (step 21: `issue_ID`, which denotes “3. The system issues login ID and password to the user” in Figure 7), Login is executed (steps 24-30: `input_ID`, denoting “1. The user inputs login ID and password to the system” in Figure 7) and then its login is successful (step 31: `logged_in`, the post condition of the Login use case). After that, the Logout use case is immediately executed (step 32: `exit`). Login and logout are iterated as a loop (step 23 → 33 → 23).

By investigating the counterexamples, we can identify where regulatory violation is in the FSM and what parts we should correct. In this example, we can find that there is regulatory violation in either Create an account, Login or Logout and that we can resolve it by inserting the obligation act “`notify(System, Purpose_of_use, User)`” to either of them. We add this act to Create an account.

4 An Example of Exemption

In this section, we introduce another example which includes a regulation denoting exemption, and it is a simplified, but actual, version of the process of money transfer at a bank. If we intend to transfer money to an account, we can take two ways; one is to ask a bank teller to do and the other is to use an automatic teller machine (ATM). In the former way, a transferer (a person who intends to transfer money) fills a form and brings it together with cash to a bank counter. The bank teller at the counter requires the transferer to show some identification documents such as photo ID card, passport, etc. to prove if the transferer is a real person who wants to transfer money or not. This process is for avoiding spoofing and voice phishing. If the identification is successful, the bank teller can perform money transfer. In the second case, the transferer transfers money from her bank account to the other account by manipulating an ATM. She inserts her cash card to the ATM and inputs her PIN code. After the valid PIN code is verified, she can specify amount of money and the account which receives it. Figure 9 shows the simplified version of the FSM of the above process and we omit the details of an ATM manipulation process like inputting PIN code etc.

Although the identification of a transferer is necessary to protect from spoofing and voice phishing, for convenience of bank customers the exemption regulation “when a person intends to transfer money cheaper than 100,000 JPY, i.e. small money, she is not identified” is enacted. We can specify this regulation as the following CTL formula.

$$\text{state} = \text{“intend to transfer small money by ATM”} \rightarrow \mathbf{EG} \neg (\text{state} = \text{“identify a transferer”})$$

By using NuSMV, we find that its result is true. Suppose that we change the FSM of Figure 9 by replacing with “identify a transferer” the destination (“transfer”) of the transition from “manipulate an ATM”. It means that the identification is necessary in

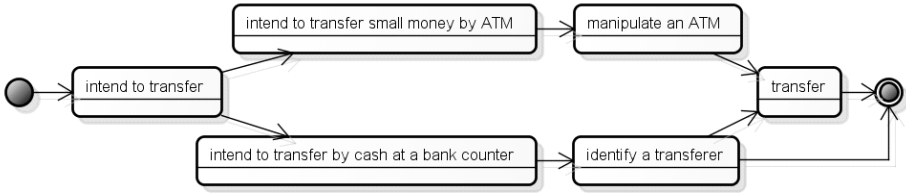


Fig. 9. Money Transfer Process

both ATM-transfer and cash-transfer. In this changed version of FSM, we get false for the above formula. However, this is not a regulatory noncompliance as mentioned in section 2.1.

Suppose the following formula on the FSM of Figure 9.

$$\text{state} = \text{“intend to transfer small money by ATM”} \rightarrow \text{AG} \neg (\text{state} = \text{“identify a transferer”})$$

It is true on the FSM. That is to say, whenever a transferer transfers small money, her identification is not performed. Although the FSM is regulatory compliant, this situation is not so good to avoid spoofing and voice phishing because she can repeat transferring small money without her identification at many times. Thus, we can consider to add to Figure 9 the transition from “manipulate an ATM” to “identify a transferer” with keeping regulatory compliance. The example mentioned in this section shows that we can find some kind of vulnerability by using additional checks related to permission and exemption regulations.

5 Related Work

The research topics related to regulatory compliance in requirements engineering area being actively focused on. The state of the art of this area and some achievements can be found in [10]. We can find many approaches to represent regulations with formal expressions [11,12,13], and many of them used classical logic as formal representations of regulations. For example, Hassan et al. used logical formula to represent regulatory statements and enterprise requirements, and Alloy analyzer to check the consistency between them [6]. Although they benefited from powerful and well-established theorem provers and model checkers, an issue on the treatment of the modalities of regulatory statements still remains. Analyzing regulatory compliance of misuse cases leads to the elicitation of a class of non-functional requirements such as security, safety, reliability etc. Deontic logic is one of the alternatives to represent the modalities of obligation and prohibition more intuitively and comprehensively [7]. However, its theorem prover or model checker has been less established yet rather than CTL [2,4]. There are some studies on applying goal-oriented analysis to regulations in order to identify the rationales of and the dependencies among regulatory statements. [3]. Although their aim does not have the same direction as ours, we can consider deeper analysis by checking regulatory compliance from the viewpoints of rationales of regulations and requirements specifications.

6 Conclusions and Future Work

This paper presents the technique to check regulatory non-compliance of a business process and an information system by using a model checking. The future work can be listed up as follows.

1. Elaborating the automated technique to translate human-friendly descriptions of FSMs including use case models and state transition diagrams into SMV FSMs. In addition, we also consider the other types of descriptions such as UML Activity Diagram and are developing its translation tool.
2. Elaborating the supporting tool and its assessment by case studies, in particular NuSMV is not so powerful to retrieve and manage counterexamples. The functions on manipulating counterexamples are significant to resolve regulatory noncompliance and the methodology how to find from the counterexamples the solutions to mitigate regulatory noncompliance should be developed.
3. Developing a supporting technique for the processes to translate correctly regulatory statements into CTL formulas. In addition, since regulatory documents include meta level descriptions such as application priority of the statements, the technique to model them should be more elaborated.
4. Considering how to deal with scalability problems on the techniques of model checking.
5. Dealing with non-functional requirements such as security.
6. Combining tightly our approach to requirements elicitation methods such as goal-oriented analysis and scenario analysis,
7. Developing the technique to manage and improve the requirements that have the potentials of regulatory noncompliance,
8. Developing metrics of measuring compliance, in fact the types and numbers of regulatory noncompliance occurrences can be considered as strength degrees of noncompliance.

References

1. Cabinet Office, Government of Japan: Act on the Protection of Personal Information (2003), <http://www5.cao.go.jp/seikatsu/kojin/foreign/act.pdf>
2. Castero, P., Maibaum, T.: A Tableaux System for Deontic Action Logic. In: van der Meyden, R., van der Torre, L. (eds.) DEON 2008. LNCS (LNAI), vol. 5076, pp. 34–48. Springer, Heidelberg (2008)
3. Darimont, R., Lemoine, M.: Goal Oriented Analysis of Regulations. In: REMO2V, CAiSE 2006 Workshop, pp. 838–844 (2006)
4. Dinesh, N., Joshi, A., Lee, I., Sokolsky, O.: Reasoning about Conditions and Exceptions to Laws in Regulatory Conformance Checking. In: van der Meyden, R., van der Torre, L. (eds.) DEON 2008. LNCS (LNAI), vol. 5076, pp. 110–124. Springer, Heidelberg (2008)
5. Eckoff, T., Sundby, N.: RECHTSSYSTEME (1997)
6. Hassan, W., Logrippo, L.: Requirements and Compliance in Legal Systems: a Logic Approach. In: Requirements Engineering and Law (RELAW 2008), pp. 40–44 (2008)
7. Jones, A., Sergot, M.: Deontic Logic in the Representation of Law: Towards a Methodology. *Artificial Intelligence and Law* 1(1), 45–64 (2004)

8. Nebut, C., Fleurey, F., Traon, Y., Jezequel, J.M.: Automatic Test Generation: A Use Case Driven Approach. *IEEE Transaction on Software Engineering* 32(3), 140–155 (2006)
9. NuSMV: A New Symbolic Model Checker (2007), <http://nusmv.fbk.eu/>
10. Otto, P., Anton, A.: Addressing Legal Requirements in Requirements Engineering. In: *Proc. of 15th IEEE International Requirements Engineering Conference*, pp. 5–14 (2007)
11. 1st International Workshop on Requirements Engineering and Law (2008), <http://www.csc2.ncsu.edu/workshops/relaw/>
12. International Workshop on Regulations Modelling and Their Validation and Verification (REMO2V), CAiSE 2006 Workshop(2006), <http://lacl.univ-paris12.fr//REMO2V/>
13. Interdisciplinary Workshop: Regulations Modelling and Deployment (2008), <http://lacl.univ-paris12.fr/REMOD08/>
14. Saeki, M., Kaiya, H.: Supporting the Elicitation of requirements Compliant with Regulations. In: Bellahsene, Z., Léonard, M. (eds.) *CAiSE 2008. LNCS*, vol. 5074, pp. 228–242. Springer, Heidelberg (2008)
15. Some, S.: Use case editor (uced), http://www.site.uottawa.ca/~ssome/Use_Case_Editor_UCEd.html
16. Whittle, J., Jayaraman, P.: Generating Hierarchical State Machines from Use Case Charts. In: *Proc. of 14th IEEE Requirements Engineering Conference (RE 2006)*, pp. 19–28 (2006)

A Decision Support Scheme for Software Process Improvement Prioritization

Arne Beckhaus^{1,4}, Lars M. Karg^{1,3}, Christian A. Graf², Michael Grottke^{2,3},
and Dirk Neumann⁴

¹ SAP Research Darmstadt, Germany

² imbus AG, Germany

³ University of Erlangen-Nuremberg, Germany

⁴ University of Freiburg, Germany

{arne.beckhaus, lars.karg}@sap.com,
christian.graf@imbus.de,
michael.grottke@wiso.uni-erlangen.de,
dirk.neumann@is.uni-freiburg.de

Abstract. Software managers pursuing process improvement initiatives are confronted with the problem of selecting potential improvements. In the field of software quality assurance, suitable decision support for prioritizing the optimization of activities according to their return on investment is not yet available. Our paper addresses this research gap. We develop a decision support scheme that facilitates the selection and prioritization of quality assurance activities. We demonstrate the scheme's applicability in three industrial case studies. By relying on the well-known COQUALMO model's characteristics and calibration data, our approach is industrially applicable with little data collection efforts.

Keywords: Software process improvement, Decision support, COQUALMO.

1 Introduction

For decades, users of software solutions have been suffering from poor solution quality [24]. Estimates for the United States show annual economic damages of billions of dollars [17]. Recently, software vendors have attempted to tackle this challenge by adapting the concepts of more mature industries, such as manufacturing [1]. This trend can also be seen in quality assurance (QA) departments.

Software defects are typically traced by means of different QA techniques. When attempting to improve the QA process, management is often confronted with the decision which technique to improve first in order to achieve the highest possible quality gains. Software Process Improvement (SPI) [2,11] promises to support quality managers in their decision-making. It is a software-industry-specific concept, defined as “changes implemented to a software process that bring about improvements” [18]. However, a light-weight, context-specific, and easy-to-apply SPI scheme has not yet been proposed. We fill this research gap by developing an SPI decision support scheme. It provides quality managers with a toolkit to prioritize improvement activities based on expected defect reduction.

Thereby, our approach relies on the Constructive Quality Model (COQUALMO) [6,7]. While COQUALMO's objective is defect prediction, our approach attempts to prioritize process improvements. We start with the same process assessment case study as COQUALMO, but we focus on the defect removal part and neglect defect introduction. Our optimization approach re-uses calibration data elicited from industry experts as provided in COQUALMO [5]. It also adapts to COQUALMO's definition of the defect removal model in order to determine the effect of the calibration constants on residual defects. Since it relies on many pre-calculated values, the approach can be applied in a highly efficient way.

The remainder of this article is structured as follows. Related work on SPI and COQUALMO is introduced in Section 2. Section 3 presents our findings in form of the proposed optimization approach. The applicability of this approach is demonstrated in three industrial case studies in Section 4. Our findings are summarized in Section 5, where we also give an outlook on future research directions.

2 Related Work

2.1 Software Process Improvement

Optimizing the processes of a company is one of the core responsibilities of its management. In business administration, various frameworks for business process improvements have been developed. Deming [9] with his Shewart cycle and Womack/Jones [25] with their Lean Thinking approach are two prominent examples. Besides targeting process improvement in general, these two frameworks have a special focus on quality.

In the software industry, SPI has been developed as an industry-specific concept [2,11]. It addresses the peculiarities of software engineering and is applicable to various kinds of software projects as commercial-of-the-shelf or in-house and custom software development.

A central part of SPI and other process improvement concepts is measurement [8,12,21,19,15,20]. Statistical process control is necessary for both transparency of the status quo and controlling the success of improvements quantitatively. Several standardized process assessment methodologies have been developed for software engineering which are usually referred to as 'Software Process Capability' or 'Maturity Models' [23].

Research on SPI often deals with change management practices and influence factors of SPI success (e.g. [22,10]). We therefore address an open research question by proposing a light-weight decision support scheme for the prioritization of quality assurance techniques according to how much they are expected to benefit from process improvement.

2.2 COQUALMO

COQUALMO is a quality model aiming at the prediction of residual defects in software development projects [6,7]. It is an extension of the project effort estimation model COCOMO II [3] which has successfully been implemented in various industrial settings.

The basic idea behind COQUALMO is to picture software artifacts as reservoirs (tanks) connected by pipes, similar to a water supply system. Defects are introduced and removed by additional pipes representing processes that may introduce or remove defects from the system—see Jones’ tank and pipe model [14] and Boehm’s defect introduction and removal model [4].

COQUALMO models the introduction and the removal of defects by two separate sub-models. The defect introduction (DI) sub-model covers the introduction of new (non-trivial) defects into the software code. It uses a subset of the cost drivers from COCOMO II [3] to derive a set of 21 parameters of defect introduction. These parameters are multiplied with the size of the software artifact. The output of the DI model is the predicted number of requirements, design, and coding defects.

The defect removal (DR) sub-model covers the identification and elimination of defects in later phases of the software project. Both sub-models must be applied in order to predict the number of residual defects after test. Since the sub-models are separate, it is possible to instantiate one without the other. We make use of this model characteristic, because in the context of our selection approach we are only interested in defect removal. In the following, we will therefore merely discuss the DR sub-model in more detail.

The defect removal sub-model relies on a multiplicative and influence factor based modeling approach. Residual defects are modeled separately for each software artifact type. Like the DI sub-model, the DR sub-model classifies defects according to the process steps in which they were created as requirements, design, and coding defects. This classification is named ‘artifact type’ in COQUALMO, and it may easily be extended. The DR sub-model considers three different defect removal techniques, which are called ‘profiles’ in COQUALMO:

- ‘Automated Analysis’ is a technique that statically checks the source code of a piece of software.
- ‘Peer Reviews’ are code inspections performed by people—hence the term ‘people reviews’ in [6].
- ‘Execution Testing and Tools’ can be seen as dynamic testing of the software product, potentially by means of dedicated testing tools.

All these techniques can help detect defects from all artifact types, although the defect may have been introduced in a much earlier phase. In COQUALMO, the number of residual defects of artifact type j is estimated as

$$ResEst_j = C_j \cdot DI_{Est_j} \cdot \prod_{i=1}^3 (1 - DRF_{ij})$$

with

j artifact type (requirements, design, or coding), $j \in \{1, 2, 3\}$;

DI_{Est_j} number of defects of artifact type j , estimated based on the DI sub-model;

C_j baseline (calibration factor) calculated from historical data;

i defect finding and removal techniques (automated analysis, peer reviews, and execution testing and tools), $i \in \{1, 2, 3\}$;

DRF_{ij} defect removal factor modeling the impact of the i -th technique on the j -th artifact.

For each defect removal technique, six maturity levels ranging from ‘very low’ over ‘low’, ‘nominal’, ‘high’, and ‘very high’ to ‘extra high’ are defined based on typical process characteristics. For each combination of artifact type j and profile i as well as each maturity level, [6] reports a value of DRF_{ij} determined by experts in a two-step Delphi study.

3 Proposed Decision Support Scheme

3.1 Differentiation from COQUALMO

The optimization approach suggested in this paper is directly derived from COQUALMO’s defect removal sub-model. Our selection approach does not rely on COQUALMO’s defect introduction sub-model. Instead, it is based on the weighting of influence factors by experts in form of the aforementioned two-step Delphi study. Besides the general characteristics of the multiplicative model for defect removal, this data set is the most important part of COQUALMO used in this study. Figure 1 illustrates how COQUALMO and our approach are linked.

In [5], the defect removal factors are provided in table format. They are reflected in the model definition in form of the DRF_{ij} factors. DRF_{ij} can be interpreted as the fraction of defects of artifact type j that can be removed by means of defect removal technique i . (Note that in this study COQUALMO’s concept of defect removal profiles is referred to as defect removal techniques.) For example, a DRF_{ij} value of 0.1 means that 10% of the defects of artifact type j can be removed via technique i .

Due to the typical constraint in an industrial case study that its findings must justify the data acquisition effort, usually only one artifact type j is taken into account. In most cases this is coding, for obvious reasons. Thus, DRF_{ij} can be reduced to DRF_i since there is no further need to distinguish between artifact types. However, this simplification of the model is not necessary, and the selection approach as derived below can easily be developed for multiple artifact types as well.

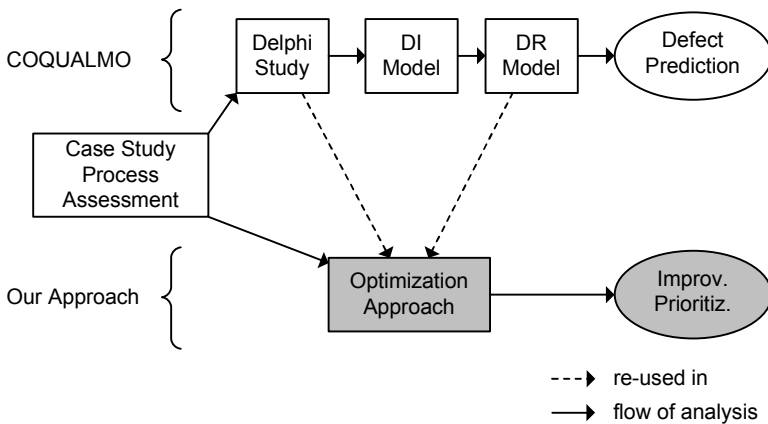


Fig. 1. Association between COQUALMO and our approach

Since DRF_i (and DRF_{ij} , respectively) are model variables that are assigned values corresponding to the maturity level m found when instantiating COQUALMO, we additionally introduce the constants $DRC_{i,m}$ where i is the defect removal technique and m is its maturity level. The values of $DRC_{i,m}$ can directly be taken from [6].

3.2 Definition

A qualitative pre-study revealed that quality managers are very interested in prioritizing improvements of the three techniques of automatic code analysis, peer reviews, and execution testing and tools. Thus, our approach extends COQUALMO in a way that provides quality management with the means to prioritize its investments into process maturity. Since the software industry is currently striving for a higher maturity of its processes [16], managers would like to know where to start in order to achieve the biggest gains. In the field of quality management, these gains can be quantified in terms of the reduction in the number of defects. In other words: If a quality manager has to choose between process improvements in the three techniques mentioned above, s/he should pick the one that is expected to yield the highest reduction in defects remaining in the software.

In fact, the constants provided in COQUALMO already contain all the information necessary for revealing the impact of moving a process to the next maturity level. Let

$$\Delta_{i,m} = DRes_{Est(i,m)} - DRes_{Est(i,m+1)}$$

be the estimated number of residual defects remaining less in an artifact when the maturity of defect removal technique i is upgraded from level m to level $m+1$. $i \in \{1, 2, 3\}$ is one of the three defect removal techniques, and $m \in \{1, \dots, 5\}$ represents one of the maturity levels except ‘extra high’, which is excluded due to the impossibility of any further improvement at this level.

Based on these concepts, we can derive

$$\text{opt}_{i,m} = \Delta_{i,m} / DRes_{Est(i,m)},$$

which is the estimated fraction by which the number of residual defects is reduced when upgrading the process from (i, m) to $(i, m+1)$. This expression can be simplified to

$$\text{opt}_{i,m} = 1 - \frac{(1 - DRC_{i,m+1})}{(1 - DRC_{i,m})},$$

where $DRC_{i,m}$ denotes the constant defect reduction factor for defect removal technique i on maturity level m for the artifact type ‘coding’ as given in [6]. $\text{opt}_{i,m}$ is the optimization index for moving the process for defect removal technique i from maturity level m to $m+1$. As shown above, it represents the estimated fraction by which the total number of remaining defects would be reduced when a specific process is moved to the next maturity level. The optimization index for the highest maturity level is undefined since, by definition, at this level further improvement is impossible.

Despite restricting our industrial case studies to the artifact type ‘coding’, the above formula can easily be extended to support COQUALMO’s other two artifact types as well. The only necessary modification is the insertion of a third index j , denoting the artifact type, to all variables.

Table 1. Optimization Matrix

	Very Low	Low	Nominal	High	Very High
Automated Analysis	10%	11%	13%	25.7%	13%
Peer Review	30%	25.7%	23.1%	33%	37%
Execution Testing & Tools	38%	32%	26.2%	29%	45%

Calculating the optimization index for all pairs (i, m) yields an optimization matrix for process maturity improvement. The optimization matrix shown in Table 1 relies exclusively on the data provided by Chulani in COQUALMO [6]. Therefore, it can be calculated without any case-study-specific data. Its entries give the percentage by which the number of residual defects in coding will be reduced when raising the process maturity by one level for the given defect removal technique.

Our selection approach will typically be instantiated by a process assessment in order to derive the current maturity levels of the three techniques. For each technique i and its current maturity level m , the value $\text{opt}_{i,m}$ is then looked up in the optimization matrix provided in Table 1. An improvement of the technique with the highest value is expected to yield the highest possible reduction of remaining defects and, consequently, the best impact on quality according to the COQUALMO model.

Note that we assume single steps of process improvement. This is due to our experience that it is very difficult to implement process improvements, and just as difficult not to fall back to the old practices over time. Our assumption is in line with the recent debate of the establishment of lean thinking and lean engineering principles in the software industry [25,16]. Recommending jumps across multiple levels, for example from ‘very low’ to ‘high’, in one step would rather stem from the opposite school of thought, namely business process re-engineering [13].

4 Case Studies

In order to demonstrate the applicability and usefulness of our approach in practice, we conducted three case studies. One of them was concerned with the software development unit at a medium-sized enterprise. The quality assurance processes of the chosen project were well established, but the project itself was rather small with a team of less than 20 developers. The other two industrial case studies were conducted at a large European business software vendor. Here, two different development projects were chosen in an innovative line of business. The team sizes were about 60 and 80 developers, respectively. We consistently number these case studies by 1 to 3, but due to confidentiality reasons we cannot provide any information on which number belongs to which development project.

Our research methodology is interview-based. For each case study, we interviewed five project members. Interviewees were chosen to cover a broad range of roles within a project. For each case study, we interviewed the quality manager responsible for the project and covered in addition at least four of the following job roles: developer, architect, tester, quality specialist, and project lead.

Table 2. Optimization Matrices for the Three Case Studies

Case Study 1	Very Low	Low	Nominal	High	Very High
Automated Analysis	10%	11%	13%	25.7% → 3.	13%
Peer Review	30%	25.7%	23.1%	33%	37% → 2.
Execution Testing & Tools	38%	32%	26.2%	29%	45% → 1.
Case Study 2	Very Low	Low	Nominal	High	Very High
Automated Analysis	10%	11% → 3.	13%	25.7%	13%
Peer Review	30%	25.7%	23.1% → 2.	33%	37%
Execution Testing & Tools	38%	32%	26.2%	29%	45% → 1.
Case Study 3	Very Low	Low	Nominal	High	Very High
Automated Analysis	10%	11%	13%	25.7%	13% → 3.
Peer Review	30%	25.7%	23.1%	33%	37% → 2.
Execution Testing & Tools	38%	32%	26.2%	29%	45% → 1.

The interview was based on a questionnaire derived from the defect profile descriptions in [5]. Our experience with two pre-studies conducted at projects 1 and 2 showed that purely questionnaire-based research yields noisy data in the form of a high variation in the process maturity estimates. This is due to difficulties on the part of the interviewees to rank their processes in an industry-wide context. We therefore asked open questions regarding the processes to come up with a first restricted set of possible maturity levels. In a next step, we provided our interview participants with examples tailored to their context in order to achieve a common understanding of the maturity levels. This methodology appeared to be a critical success factor in low-effort process maturity assessments, since our default explanations of the different maturity levels were often only understood after providing context-specific details and examples.

An alternative data acquisition method would have been a CMMI appraisal. However, its effort is very high compared to our approach and would not have been justifiable in our setup. Nevertheless, CMMI has gained popularity, and it is recommended to check for existing appraisals prior to conducting new interviews. Since it provides valuable input for other business decisions, a new appraisal might also be worth the effort when combining our approach with others.

Participants of our case study praised our efficient data acquisition methodology and voluntarily asked us to repeat this assessment every six months for benchmarking and controlling purposes. However, a low-effort questionnaire-based process assessment methodology may introduce a bias into the data acquisition. We encourage considerable evaluation of the data quality needs of a usage scenario prior to conducting a survey.

According to the selection scheme discussed in Section 3, we highlight the maturity levels of the defect removal techniques in our optimization matrices in Table 2. The improvement rank of the test activities is given to the right of the arrows. For example, in case study 1, it would be best to invest into an improvement of execution testing and tools. Raising the process maturity level from very high to extra high is expected

to yield a reduction of residual defects by 45%. Second-best is the improvement of peer reviews with an improvement factor of 37%. The improvement of the automated analysis technique from high to very high process maturity ranks third with an estimated improvement of 25.7%.

After conducting our three case studies, we asked the participating quality managers whether or not they agree with the prioritization derived by our approach. This cross-check was successful, and most managers accepted the finding that test execution environments are an attractive area for attaining high returns on investment in improvement initiatives.

5 Conclusions

In this paper, we presented a decision support approach to prioritize three different quality assurance techniques for selection in improvement projects. It is based on the multiplicative model definition of COQUALMO, as well as its calibration data gathered in the course of a two-step Delphi study [5]. Our approach facilitates the current advancement of the software industry in the form of managed, lean processes. Quality managers are able to prioritize process improvements based on their expected effect on quality in terms of residual defects. The approach can be instantiated with low effort due to the re-use of COQUALMO constants. It is also context-specific due to relying on process assessments. Our approach's applicability has successfully been demonstrated in three industrial case studies with a medium-sized enterprise and a global player in the software industry.

Future research is needed in order to also quantify the investment needed to raise process maturity levels. Once these data are available, quality managers are able to economically trade off between the expected quality enhancement yield of an improvement initiative on the one hand and its costs on the other hand. Additionally, our approach should be validated by conducting repetitive case studies after processes have been improved and lifted to higher maturity levels. In this way, the assumptions concerning defect reductions inherent in the Delphi-study calibration data of COQUALMO can be cross-checked and possibly refined.

Acknowledgements. Parts of the work presented in this paper have been funded by the German Federal Ministry of Education and Research (grants 01ISF08A and 01ISF08B).

References

1. Antony, J., Fergusson, C.: Six sigma in the software industry: results from a pilot study. *Managerial Auditing Journal* 19, 1025–1032 (2004)
2. Basili, V., Caldiera, G.: Improve software quality by reusing knowledge and experience. *Sloan Management Review* 37(1), 55–64 (1995)
3. Boehm, B., Abts, C., Brown, A., Chulani, S., Clark, B., Horowitz, E., Madachy, R., Riefer, D., Steece, B.: *Software Cost Estimation with COCOMO II*. Prentice-Hall, Englewood Cliffs (2000)
4. Boehm, B.W.: *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs (1981)

5. Chulani, S., Boehm, B.: Modeling software defect introduction and removal: COQUALMO (CONstructive QUALity MOdel). Tech. rep., Technical Report USC-CSE-99-510, University of Southern California, Center for Software Engineering (1999)
6. Chulani, S.: COQUALMO (CONstructive QUALity MOdel) a software defect density prediction model. In: Kusters, R., Cowderoy, A., Heemstra, F., van Veenendaal, E. (eds.) *Project Control for Software Quality*, Shaker Publishing, Ithaca (1999)
7. Chulani, S., Steece, B.M., Boehm, B.: Case Studies in Reliability and Maintenance. In: *Determining Software Quality Using COQUALMO*, pp. 293–311. Wiley, Chichester (2003)
8. DeMarco, T.: *Controlling Software Projects: Management, Measurement and Estimation*. Yourdon Press, New York (1982)
9. Deming, W.E.: *Out of the Crisis*. MIT Press, Cambridge (2000)
10. Dyba, T.: An empirical investigation of the key factors for success in software process improvement. *IEEE Transactions on Software Engineering* 31(5), 410–424 (2005)
11. El Emam, K., Drouin, J.N., Melo, W. (eds.): *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*. CS Press (1998)
12. Fenton, N., Pfleeger, S.: *Software Metrics: A Rigorous and Practical Approach*. Int'l Thomson Computer Press, London (1996)
13. Hammer, M., Champy, J.: *Reengineering the Corporation. A Manifesto for Business Revolution*. Collins Business (2003)
14. Jones, C.: Programming defect removal. In: *Proceedings, GUIDE 40* (1975)
15. Jones, C.: *Applied Software Measurement: Global Analysis of Productivity and Quality*, 3rd edn. McGraw-Hill, New York (2008)
16. Middleton, P., Sutton, J.: *Lean Software Strategies*. Productivity Press (2005)
17. NIST: The economic impacts of inadequate infrastructure for software quality (2002)
18. Olson, T.G., Humphrey, W.S., Kitson, D.: Conducting SEI-assisted software process assessments. Tech. rep., Carnegie Mellon University, Technical Report CMU/SEI-89-TR-7, Pittsburgh (1989)
19. Rifkin, S.: What makes measuring software so hard? *IEEE Software* 18(3), 41–45 (2001)
20. Sahraoui, H., Briand, L.C., Guhneuc, Y.G., Beaurepaire, O.: Investigating the impact of a measurement program on software quality. *Information & Software Technology* 52(9), 923–933 (2010)
21. van Solingen, R., Berghout, E.: *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill, London (1999)
22. Stelzer, D., Mellis, W.: Success factors of organizational change in software process improvement. *Software Process Improvement and Practice* 4(4), 227–250 (1998)
23. Von Wangenheim, C.G., Hauck, J.C.R., Zoucas, A., Salviano, C.F., McCaffery, F., Shull, F.: Creating software process capability/maturity models. *IEEE Software* 27(4), 92–94 (2010)
24. Whittaker, J.A., Voas, J.M.: 50 years of software: Key principles for quality. *IT Pro*, 28–35 (November/December 2002)
25. Womack, J.P., Jones, D.T.: *Lean Thinking*, 2nd edn. Free Press, New York (2003)

Development of a Family of Personalized Mobile Communicators

Miguel A. Laguna and Bruno González-Baixauli

GIRO Research Group, University of Valladolid
Campus M. Delibes, 47011 Valladolid, Spain
{mlaguna, bbaixauli}@infor.uva.es

Abstract. People with communication problems can use personal communicators as a low-cost help in their everyday life. The diversity of individual situations has guided us towards a solution based on the software product line paradigm. Multiple options can be easily incorporated to each product, allowing the adequate customization of the final application to the disability of each concrete person. Software product lines are a proven development paradigm in industrial environment but its application in small organizations is not easy. Our approach uses the UML package merge mechanism to manage the variability in the product line requirement, design and implementation models. The structure of the feature models is directly reflected in the relationships between packages in the architectural models, so that the traceability of configuration decisions is straightforward. A similar strategy is applied at the implementation level, using packages of partial classes. The combination of these techniques and the conventional IDE tools make the developments of product lines in small organizations easier as it removes the need for specialized tools and personnel. This article reports the successful experience on the development of a family of personalized communicators as a software product line representative of the mobile systems domain.

Keywords: Software product line, Feature model, Mobile system, communicator.

1 Introduction

Quality of life is a general aspiration of people, especially associated to subjective feelings about general health, communication skills, and mobility. On the other hand, wireless and mobile devices are being used for multiple purposes with an increasing list of applicability domains. In particular, people with communication problems can use mobile communicators as a low-cost help in their everyday life. However, the diversity of individual situations is a problem that has guided us towards a solution based on the software product line paradigm. Using this approach, multiple options can be easily incorporated to each product, allowing the adequate customization of the final application to the disability of each concrete person.

Software product lines (SPL) are a proven reuse approach in industrial environments, due to the combination of a systematic development and the reuse of coarse-grained components that include the common and variable parts of the product line

[2]. However, this approach is complex and requires a great effort by the companies that take it on. The research we carry out in the GIRO research group¹ aims to simplify the change from a conventional development process into one that benefits from the product line advantages in small and medium enterprises (SME) or organizations. For this reason, we have proposed, among other initiatives, an adaptation of the Unified Process to include specific techniques of Product Line Engineering in a process parallel to Application Engineering [11].

As specific SPL development techniques, we must pay special attention to the variability and traceability aspects at each abstraction level. We need models that represent the product line and a mechanism to obtain the configuration of features that represent the best combination of variants for a specific application. Additionally, we must connect the optional features with the related variation points of the architectural models that implement the product line through traceability links. There is wide agreement about using a model that shows, in an explicit and hierarchical way, the variability by means of a feature model in some of their multiple versions like FODA [10] or FORM [9]. FODA features are nodes of a tree, related by various types of edges (Figure 1). The tree root is called the root feature, or concept. The edges are used to decompose this concept into more detailed features. There are AND, X-OR and optional decompositions. Several extensions have been proposed, using directed acyclic graphs instead of simple trees or changing the visual syntax.

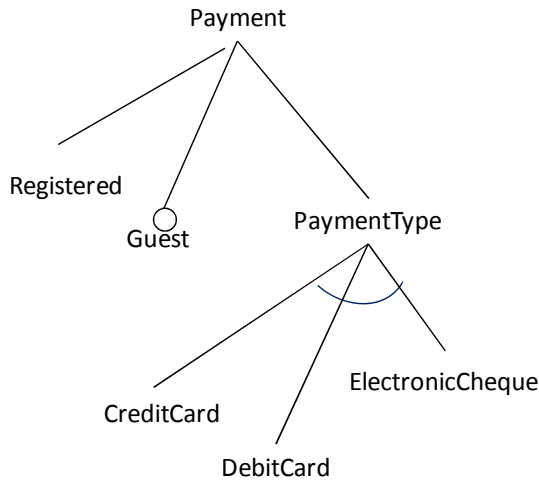


Fig. 1. A simple FODA feature diagram

We have also proposed the use of the goal and soft-goal concepts [16] for the analysis and definition of the variability in product lines. In fact, we have built an initial prototype that permits the optimum set of features to be selected with respect to the desires of the users, expressed as a set of goals and soft-goals with different priorities [6].

¹ <http://www.giro.infor.uva.es>

The second aspect of the problem focuses on the connection of the feature model with the design of the solution or product line architecture, usually implemented by an object-oriented framework. This explicit connection allows the automatic instantiation of the domain framework in each specific application. In a previous work [12], we proposed the UML 2 package merge mechanism to orthogonally represent the SPL architectural variations, their relationship with the optional features and finally, using partial class packages, with the structure of the implementation code.

We planned, as a continuation of this work, to test the proposal in realistic situations. Our group has agreements with associations of handicapped people with the aim of developing useful tools for people with several types of disabilities. This background has guided the selection of the application domains. This article is a report of the practical experiences with these techniques in the development of a product line of personalized communicators for people with disabilities, based on mobile devices, typically personal digital assistants (PDA).

A distinctive characteristic is the use of conventional CASE and IDE tools. This is a pre-requisite imposed by the general objective of our approach: to simplify the adoption of the product line paradigm by SMEs. In particular, we have used .NET and MS Visual Studio as development platforms. The personnel involved vary from granted and volunteer postgraduate students to undergraduates finishing their term projects, but they are not specialists in SPL development.

The rest of the article is organized as follows: Section 2 briefly presents the proposed techniques, based on the package merge relationship of UML 2 and the partial class mechanism. Section 3 is devoted to the description of the case study. In Section 4, the related work is analyzed and, finally, Section 5 concludes the article, states some lessons learned and outlines future work.

2 Seamless SPL Development

Each concrete system in a product line is derived from a complete architecture, selecting or not the optional parts, with respect to the particular functional and non-functional user requirements. This activity is basically a selection process that yields a feature sub-model. This sub-model, through traceability relationships, guides the composition of the code modules. The key aspect of the process is the traceability from features to design and from design to implementation. This traceability is not easily managed for several reasons [15]. On the one hand, an optional feature can be related to several elements in a UML model and vice versa. We must therefore assign the traceability relationship between elements of the two levels with a “many-to-many” multiplicity. This fact quickly complicates the global model, making it poorly scalable. The second problem is summarized in the fact that the same basic modeling mechanisms of variability (the specialization in class diagrams or the <<extend>> relationship of the use cases diagrams) are used to express two variability levels: the design of the product line architecture and the design of a specific application that also has variations (for example two valid and alternative payment forms within a sales system).

The solution to this problem has been achieved by modifying or adapting the UML structural and behavioral models, moving from the standard (see the references of the related work Section). In our approach, one of the initial restrictions imposed was to maintain unchanged the UML meta-model, in order to use conventional CASE tools to model the product line. Other obligations were:

- a) The technique must allow the location, at one point on the model, of all the variations associated to each optional feature to facilitate the management of the traceability.
- b) The technique must separate the SPL from the intrinsic variability of the specific applications.
- c) The selected mechanism must have continuity with the implementation models (“seamless development”).

To achieve these objectives, we express the variability of UML models using the package merge mechanism, defined in the UML 2 infrastructure meta-model and used in an exhaustive way in the definition of UML 2 [14].

The package merge mechanism adds details to the models in an incremental way. The <<merge>> dependence is defined as a relationship between two packages that indicates that the contents of both are combined. It is very similar to generalization and is used when elements in different packages have the same name and represent the same concept, beginning with a common base. Selecting the desired packages, it is possible to obtain a tailored definition from among all the possible ones. Even though, in this work, we focus on class diagrams, the mechanism can be extended to any UML model, in particular use cases and sequence diagrams [14].

This mechanism permits a clear traceability between feature and UML models to be established. The application to our problem consists in associating a package to each optional feature, so that all the necessary changes in the model remain located (maintaining the UML meta-model unchanged and separating both variability levels).

The package model is hierarchical, reflecting the feature model structure. Considering each pair of related packages recursively, the base package can be included or not in each specific product, but the dependent package can only be included if the base package is also selected. This is exactly how experts decide which features are included or not during the configuration process, and is directly reflected in the final product configuration of packages. Therefore, the application to SPL consists of building the architectural model (including structural –class diagrams-, behavioral -use cases-, and dynamic –interaction diagram- models) starting from a base package that gathers the common SPL aspects. Then, each variability point detected in the feature model originates a package, connected through a <<merge>> relationship with its parent package. These packages will be combined or not, when each product is derived, according to the selected feature configuration. Figure 2 shows an example of application in the e-commerce domain.

Additionally, using partial classes organized in packages, a direct correspondence between design and code can be established. The use of partial classes is a way of managing variability. The aim is to maintain a one-to-one correspondence from features to design and from design to implementation. As an added value, the package structure of the code for each final product of the SPL can be obtained automatically (and passed to the compiler) from the features configuration [12].

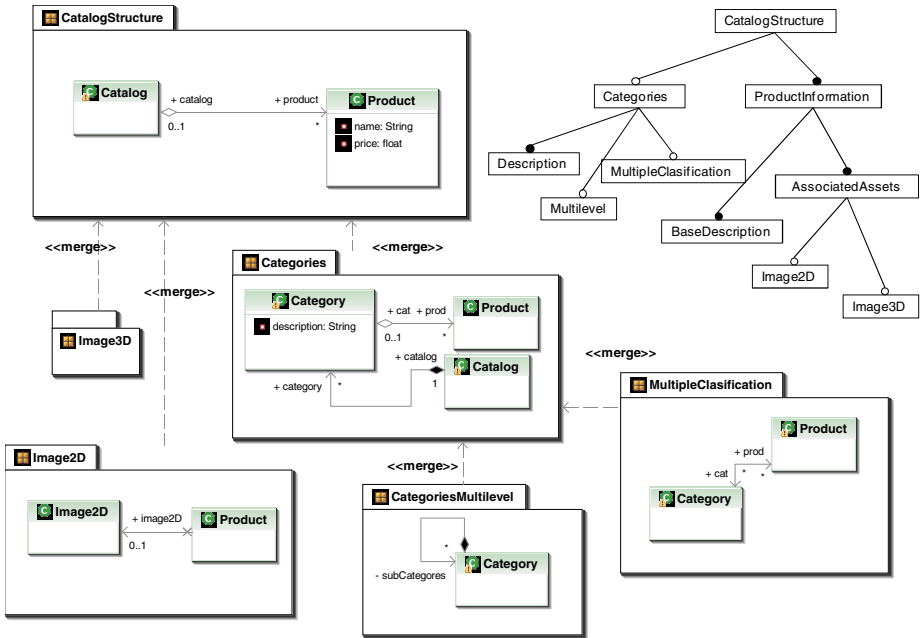


Fig. 2. A partial Feature Model and a possible UML design

3 Case Study: Communicators for People with Disabilities

The case study is not very ambitious if we judge it by the number of considered variations but presents interesting problems, due to the constraints imposed by the specificity of mobile device development.

The domain analysis has been accomplished starting from the experience with several PDA systems developed in our laboratory. Each one of these originally solved the particular problem of a concrete person with some degree of disability. These systems have been built in collaboration with *Asprona*, a Spanish association that maintains several schools specialized in children with medium/severe disabilities of several types. The main utility of these communicators is that people with different degrees of

Table 1. Comparison of different writing methods

Writing method	Speed required	Capacity	Learning
Swept	Very slow	Very little	Very little
Sweep (with sound)	Very slow	Very little	Very little
Sweep (groups)	Slow	Very little	Little
Diagonals	Middle	Little	High
Repeated pulsations	Middle	Middle	Middle
Databases	Rapid	Middle	Middle
Traits	Very rapide	High	High
Grouped characters	Rapid	Middle	Middle
Vowels	Rapid	Middle	High

disability can compose messages using text (selecting the different characters as in a keyboard) or images (that represent different concepts) in a suitable (and usually mobile) device. The suitable methods are compared in Table 1. Once composed, the device can reproduce the message using a text-to-speech conversion (or send it to another device). The product line approach has a clear intention: separate the common parts of these systems from the specialized ones, developing these parts as optional packages. As an immediate result, we have multiplied the number of different available variants.

3.1 Feature Analysis

All the final applications must reproduce the text composed by the user. But, due to the different abilities of the users, it is necessary to consider different writing methods, adapted to each type of disability. For example, if the user is not capable of clicking a button, it is necessary to use a sweeping method. We have considered several (textual and image based) writing methods. Some of them are the following:

- Grouped characters method: the main screen shows several groups of characters (AÁBCDE, ÉFGHIÍ, etc.). Selecting a group enables another screen, where the characters of this group appear redistributed, one per cell. The selection of one of them results in that character being added to the text.
- Vowels method: similar to the previous method, but the vowels are represented in independent cells on the main screen, generally reducing the number of pulsations.
- Categories method: the categories of characters (consonants, vowels and numbers) are shown in the initial screen.

Each of the evaluated methods has advantages and inconveniences for people with different degrees and types of disabilities, as shown in Table 1. Using the table as a guide, and adding some complementary aspects such as color management, phrases persistence, etc., the feature model of Figure 3 has been defined. For legibility reasons, the original graphical tree format is depicted in a compact alternative representation.

The feature model has been defined with the Feature Modeling Tool (FMT, available at GIRO site²), developed in our group with Microsoft DSL tools as an add-in of Visual Studio. The DSL Tools is an SDK for MS Visual Studio that allows domain-specific languages to be defined and modeling tools to be generated for these languages. We used the Microsoft DSL tools to build the graphical tool, and then we created the associated editors and utilities using the .NET native facilities. Figure 4 shows a general picture of FMT with its four views: the graphical modeling tool (Design view), Solution Explorer (the native Visual Studio view), Configuration view, and the auxiliary tree view (or feature Model Explorer). The main window is the Design view, where feature models are visually built, including *requires* and *mutex* dependencies. The graphical view is complemented by the Model Explorer view, visually identical to the format used by the *fmp* plug-in (see Figure 3 details). The lower left view is the Configuration panel, where the product engineer can select those features that s/he wants to be included in each SPL specific product. The tool

² <http://www.giro.infor.uva.es/FeatureTool.html>

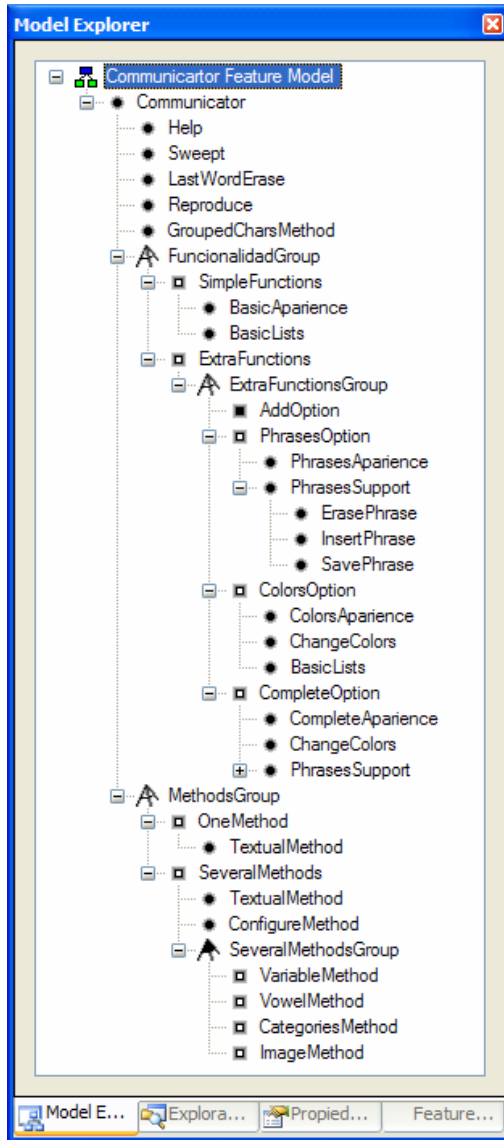


Fig. 3. Feature model of the communicator product line

itself is responsible for checking the multiplicities and *requires/mutex* constraints. Finally, the last view (upper right) is the view of the SPL as a development project (a Visual Studio *solution*). This view automatically changes with the SPL configuration: If a feature is selected in the configuration panel, the linked package is added to the solution. The modeling tool is completed with the package generation and configuration utilities, as explained in the previous Section.

According to the model of Figure 3, each final product can incorporate several writing methods, but all the systems will have at least the grouped characters method. For this reason, the right structure of the feature model has two main alternative branches. If more than a writing method is selected, the exchange from a writing method to another must be allowed. Then the *Configure writing method* is mandatory. This type of relationship between features enriches the model but must be carefully registered in the feature model.

3.2 Product Line Design

In Figure 4, some of the packages and classes that make up the product line can be appreciated inside the original Visual Studio solution explorer (upper right panel of the image). Each package contains internally a set of partial classes that the compiler will integrate if the package is selected (i.e., if the optional feature is selected in the bottom left configuration panel of Figure 4).

In this type of applications the need for persistence is limited (only preferences and customer phrases are saved), but interface design requires a greater effort, due to the

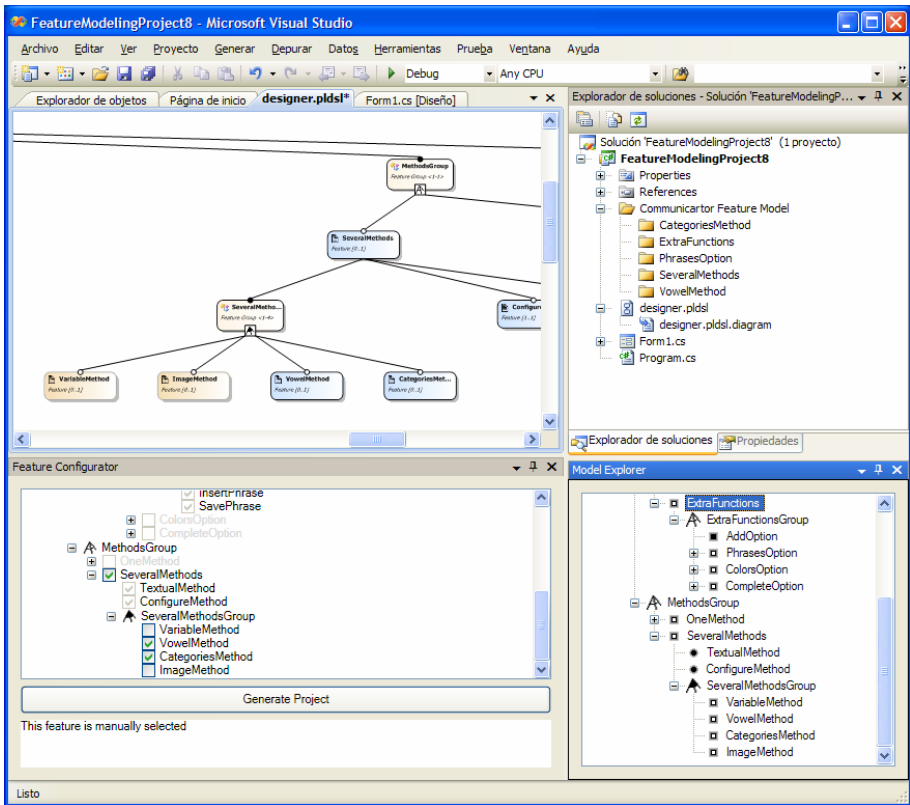


Fig. 4. Feature model tool and the communicator product line, solution, configuration and model views

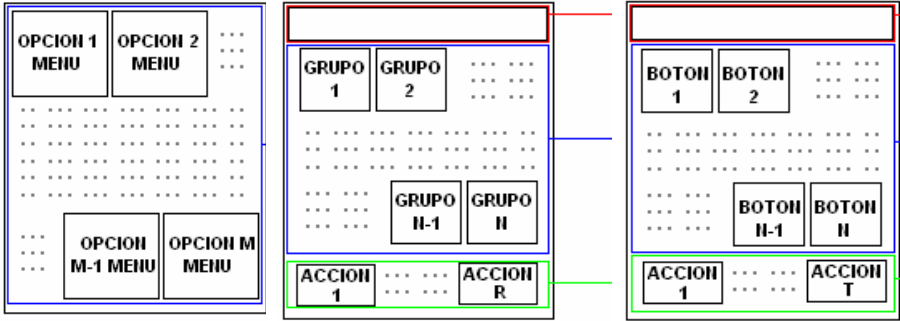


Fig. 5. Structure of the configurable user interface of the communicator product line

limitations of the visualization screen. To deal with these limitations, XML and XSD files that define the elements of the interface are used. The variable parts are included in the different packages.

In Figure 5, the design of the user interface is shown in a diagrammatical way. The size and number of buttons are variable and the selected configuration indicates the number of buttons on each screen, as well as size, position, functionality and appearance. For example, if the *Textual* package is selected, the *principal_cat.xml* file defines the welcome screen and creates the communicator according to the selected writing method. Each .XML file is located in the package that includes its related functionality. Previous to compilation, the required .XML files are combined (in a way similar to the C# partial classes facility) using a DATASET based mechanism.

3.3 Product Line Implementation

At implementation level, partial classes and conditional compilation have been used. The strategy consists of using the same code structure in all the cases, as a template. The *Base* package contains a main class *Program.cs*, where the code that loads the common part of the product line is included. The optional packages contain a class *Program.cs* with the methods that add the package, executed from the source code through conditional compilation. For example, the package *CompleteOptions* has a class with the methods that add color details and the management of predefined phrases, updating the menu with the new options.

One of the components that the product line must include is the text-to-speech utility. In spite of the available commercial and open-source applications, the limitations of the mobile platforms have forced to an *ad-hoc* solution, developing a simple syllabic synthesizer, with the collaboration of the students who lend their voices.

The product line includes eight thoroughly functional applications, compiled from different package combinations (some examples can be appreciated in Figure 6). Pending integration is an optional feature already implemented that will allow wireless and SMS based communication with a desktop computer.

A first working prototype has been delivered to the Asprona association specially configured for a person with speech problems but good manual coordination, as a result of a traffic accident. In this case, the grouped characters method is a good election. The use of the system, fixed to his wheel chair, is helping him to get a greater level of autonomy.

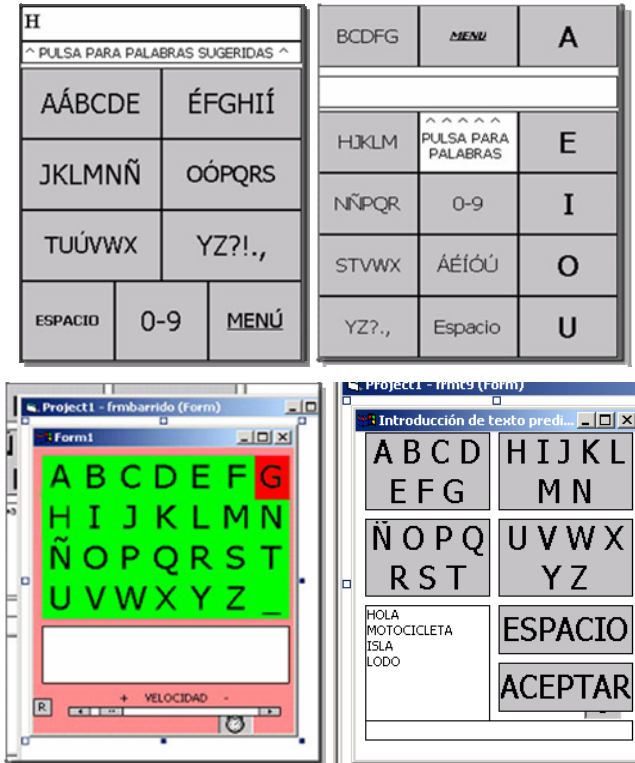


Fig. 6. Grouped characters, vowels, and sweep modules of the communicator product line

4 Related Work

Though there are many projects that describe variability management mechanisms in terms of requirements and designs, few of them include implementation details. Different authors have proposed explicitly representing the variation points adding annotations or changing the essence of UML. For example, Von der Maßen et al. proposed using new relationships ("option" and "alternative") and the consequent extension of the UML meta-model [13]. John & Muthig suggest the application of use case templates to represent the variability in product lines, using stereotypes [8], though they do not distinguish between optional variants, alternative or obligatory.

On the other hand, Halman and Pohl defend the modification of use case models to orthogonally represent the variation points (using a triangle symbol with different annotations) [7]. As for structural models, either the mechanisms of UML are used directly (through the specialization relationship, the association multiplicity, etc.) or the models are explicitly annotated using stereotypes. The work of Gomaa is an example of this approach, since it uses the stereotypes <<kernel>>, <<optional>> and <<variant>> (corresponding to obligatory, optional, and variant classes) [5]. Similarly, Clauß proposes a set of stereotypes to express the variability in the architecture



Fig. 7. A final prototype, configured using the grouped characters method

models: <<optional>>, <<variationPoint>> and <<variant>> stereotypes designate, respectively, optional, variation points (and its sub-classes), and variant classes [3]. Though this type of approximations permits the evolution of the variability to be traced at the different levels, they do not solve the requirement of a one-to-one correspondence between the different models.

Another solution proposed by Czarnecki in [4], consists of annotating the UML models with presence conditions, so that each optional feature is reflected in one or, more realistically, several parts of a diagram (perhaps a class, an association, an attribute, etc. or a combination of elements). This technique does not artificially limit the representation of a variant with a unique element and even the color code helps to emphasize the implications of choosing a certain option. However, this visual help is not scalable when more than a dozen variants are handled. In all these approaches, the modification of the UML meta-model (or at least the use of stereotypes) is required.

A completely different approach, focused on implementation instead of requirements or design, is the Feature Oriented Programming (FOP) paradigm [1]. The optional features are implemented as increments (refinements) in a java-like language. Starting from a base class, these increments are combined using a set of tools, provided with the AHEAD tool suite. Other commercial tools, such as Big-Lever Gears or Pure-Variants offer similar functionalities.

Though these solutions are valid, the learning of new modeling or implementation techniques and the need of specialized CASE and IDE tools represent barriers for the adoption of the approach of product lines in many organizations; we therefore believe that the solution presented here improves the abovementioned proposals.

5 Conclusions

In this work the viability of a product line development approach, based on the package merge and partial class mechanisms, has been shown. The use of the proposed mechanisms enables the automated generation of each product from the features configuration. Furthermore, the use of conventional CASE and IDE tools can simplify the adoption of this paradigm, avoiding the necessity of specific tools and techniques as in previous alternatives.

The approach has been successfully applied to the design and implementation of a product line in the domain of communicators for people with disabilities, and implemented with mobile devices.

Current work includes the development of other product lines with industrial or social interest, and the enrichment of the communicator study. In this case, the objective is to evaluate the scalability of the proposal as the optional features increase (which implies an exponential increase in the number of final products). On the other hand, the experience with this type of mobile platform is being used in other domains that combine information capture through PDAs and smart phones with delivery to a central system, configured as a set of Web services. An example of this is a recently launched product line project for monitoring health parameters (such as heart rate, temperature, etc.) in the context of a senior citizen residence, using a combination of wireless sensors and mobile devices. The utility of the product line approach in these domains is evident, as the variety of sensors, parameters, alarm signals, and visualization aspects in the central computer is potentially unlimited.

Acknowledgements. This work has been founded by the Junta de Castilla y León (VA-018A07 project) and Spanish MICIINN (TIN2008-05675). We also recognize the collaboration of the ASPRONA association, and the students involved in the development of these product lines.

References

1. Batory, D., Sarvela, J.N., Rauschmayer, A.: Scaling Step-Wise Refinement., IEEE TSE (2004)
2. Bosch, J.: Design & Use of Software Architectures. In: Adopting and Evolving a Product-Line Approach, Addison-Wesley, Reading (2000)
3. Clauß, M.: Generic modeling using UML extensions for variability. In: Workshop on Domain Specific Visual Languages at OOPSLA (2001)
4. Czarnecki, K., Antkiewicz, M.: Mapping Features to models: a template approach based on superimposed variants. In: Glück, R., Lowry, M. (eds.) GPCE 2005. LNCS, vol. 3676, pp. 422–437. Springer, Heidelberg (2005)
5. Gomaa, H.: Object Oriented Analysis and Modeling for Families of Systems with UML. In: Frakes, W.B. (ed.) ICSR 2000. LNCS, vol. 1844, pp. 89–99. Springer, Heidelberg (2000)
6. González-Baixauli, B., Leite, J., Mylopoulos, J.: Visual Variability Analysis with Goal Models. In: Proc. of the RE 2004, pp. 198–207 (2004)
7. Halmans, G., Pohl, K.: Communicating the Variability of a Software-Product Family to Customers. Journal of Software and Systems Modeling, 15–36 (2003)

8. John, I., Muthig, D.: Tailoring Use Cases for product line Modeling. In: Proceedings of the International Workshop on Requirements Engineering for product lines 2002 (REPL 2002). Technical Report: ALR-2002-033, AVAYA labs (2002)
9. Kang, K.C., Kim, S., Lee, J., Kim, K.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. *Annals of Software Engineering*, 143–168 (1998)
10. Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute (Carnegie Mellon), Pittsburgh, PA 15213 (1990)
11. Laguna, M.A., González, B., López, O., García, F.J.: Introducing Systematic Reuse in Mainstream Software Process. In: *IEEE Proceedings of EUROMICRO 2003*, pp. 351–358 (2003)
12. Laguna, M.A., González-Baixauli, B., Marqués, J.M.: Seamless Development of Software Product Lines: Feature Models to UML Traceability. In: *GPCE 2007(2007)*
13. von Massen, T., Lichter, H.: RequiLine: A Requirements Engineering Tool for Software product lines. In: van der Linden, F.J. (ed.) *PFE 2003*. LNCS, vol. 3014, pp. 168–180. Springer, Heidelberg (2004)
14. Object Management Group. Unified modeling language specification version 2.0: Infrastructure. Technical Report ptc/03-09-15. OMG (2003)
15. Sochos, P., Philippow, I., Riebisch, M.: Feature-oriented development of software product lines: mapping feature models to the architecture. In: Weske, M., Liggesmeyer, P. (eds.) *NODe 2004*. LNCS, vol. 3263, pp. 138–152. Springer, Heidelberg (2004)
16. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: *Proceedings of the 5 IEEE Int. Symp. on Requirements Engineering*, pp. 249–262 (2001)

Reverse Generics: Parametrization after the Fact

Alexandre Bergel¹ and Lorenzo Bettini²

¹ PLEIAD Laboratory, Computer Science Department (DCC)
University of Chile, Santiago, Chile

² Dipartimento di Informatica, Università di Torino, Italy
<http://www.bergel.eu>
<http://www.di.unito.it/~bettini>

Abstract. By abstracting over types, generic programming enables one to write code that is independent from specific data type implementation. This style is supported by most mainstream languages, including C++ with templates and Java with generics. If some code is not designed in a generic way from the start, a major effort is required to convert this code to use generic types. This conversion is manually realized which is known to be tedious and error-prone.

We propose *Reverse Generics*, a general linguistic mechanism to define a generic class from a non-generic class. For a given set of types, a generic is formed by unbinding static dependencies contained in these types. This generalization and generic type instantiation may be done incrementally. This paper studies the possible application of this linguistic mechanism to C++ and Java and, in particular, it reviews limitations of Java generics against our proposal.

1 Introduction

The concept of generic programming [8], which has characterized functional programming for several decades, appeared in mainstream programming object-oriented languages such as C++, only in the late 80s, where it motivated from the beginning the design of the Standard Template Library (STL) [17,18,5]. Generic programming was not available in the first versions of Java, and this limited code reuse, by forcing programmers to resort to unsafe operations, i.e., type casts. *Generics* are a feature of the Java 1.5 programming language. It enables the creation of reusable parameterized classes while guaranteeing type safety.

In spite of the limitations of Java generics, type parameterization allows the programmer to get rid of most type down-casts they were forced to use before Java generics; this also is reflected in part of the standard Java library which is now generic. However, much pre-1.5 Java code still needs to be upgraded to use generics. For example, a quick analysis on the AWT Java library shows that some classes perform more than 100 down-casts and up-casts and 70 uses of `instanceof`. This examination reveals that in many places the amount of up-casting subsequent down-casting that is used almost makes the programs behave like dynamically typed code.

Note, that the need to make existing code generic may arise also in languages where generic types were already available. In particular, either a library or a framework is designed in a type parametric way from the very beginning, or the “conversion” must be

done manually, possibly breaking existing code. Several proposals have been made that promote an automatic conversion from non-generic code into generic code [9,22,14]. These approaches involve reverse-engineering techniques that infer the potential type candidates to be turned into parameters. This paper presents a different approach: instead of relying on the programming environment to pick up type parameters, a programmer may create a generic class starting from a non-generic one, by using a proper language construct. To our knowledge, no previous attempt to offer a language built-in mechanism to generalize classes has been proposed.

We propose to extend the way generics are expressed by defining a generic type (class or interface) from a non-generic type definition. Our approach consists of an extension to be applied to an existing object-oriented programming language. However, in this paper, we will investigate the possible application of this extension to Java and C++. We will also demonstrate how the implementation of generic types in the language affects the usability of this new linguistic extension. With this extension, programmers can create generic classes and interfaces starting from existing classes and interfaces by specifying the types that need to be turned into generic parameters. We call this linguistic mechanism *Reverse Generics*.

This approach is different from the above mentioned refactoring approaches since in our context there will be no refactored code: the starting class will continue to exist after it is used to create its generic version. However, we believe that the refactoring techniques in the literature could formulate a refactoring based on our reverse generics to actually implement the refactoring.

The paper is organized as follows. Section 2 presents and illustrates *Reverse Generics*. Section 3 enumerates several typing issues due to the Java type system with respect to generic programming. Section 5 presents briefly related work and Section 6 concludes and presents future work.

2 Reverse Generics

Reverse Generics is an extension for object-oriented programming languages that enables a generic class to be created starting from an existing class, and a generic interface from an existing interface. Some of the static type references contained in the original class or interface are transformed into type parameters in the resulting generic version. The process of obtaining a generic class from a class definition is called *generalization*. We consider *generalization* as the dual of the *instantiation* operation. We refer to *unbinding* a type when references of this type contained in the original class definition are not contained in the generic.

Given a class name, `ClassName`, and a type name, `TypeName`, the generic version of a class is denoted by the following syntactic form:

```
ClassName>TypeName<
```

All references to `TypeName` contained in the class `ClassName` are abstracted. A name is associated to the abstract type in order to be concretized later on. Several type names may be abstracted using the following writing:

```
ClassName>TypeName1, ..., TypeNamen<
```

The resulting generic class should then be assigned to a class definition, which depends on the actual programming language (and in particular on its syntax for parameterized types); thus, in Java we would write:

```
class MyGenericClass<T extends TypeName> =
    ClassName>TypeName<;
```

In C++ we would instead write

```
template<typename T>
class MyGenericClass<T> =
    ClassName>TypeName<;
```

The class resulting from a reverse generic should be intended as a standard class in the underlying language. We could simply instantiate the type parameters of a generic class and then create an object, e.g.,

```
new MyGenericClass<MyTypeName>();
```

However, there might be cases (e.g., when using partial instantiation, Section 2) where it is useful to simply instantiate a generic class and assign it another class name; thus, we also consider this syntax:

```
class MyClass = MyGenericClass<MyTypeName>;
```

In the following, we informally describe and illustrate Reverse Generics with several examples resulting from an experiment we conducted on the AWT graphical user interface Java library. The same mechanism may be applied to C++ templates.

Class Generalization. The class `EventQueue` is a platform-independent class that queues events. It relies on `AWTEvent`, the AWT definition of event. The code below is an excerpt of the class `EventQueue`:

```
public class EventQueue {
    private synchronized AWTEvent getCurrentEventImpl() {
        return (Thread.currentThread() == dispatchThread)
            ? ((AWTEvent)currentEvent.get()): null;
    }
    public AWTEvent getNextEvent()
        throws InterruptedException {
        ...
    }
    public void postEvent(AWTEvent theEvent) {
        ...
        boolean notifyID = (theEvent.getID() == this.waitForID);
        ...
    }
    ...
}
```

In some situations, the `EventQueue` class may have to be used with one kind of event, say `KeyEvent`. This will significantly reduce the number of runtime downcasts and ensure type safety when such a queue has to be used.

By using reverse generics we can define the generic class `GEventQueue<T extends AWTEvent>` from the non-generic class `EventQueue` as follows:

```
class GEventQueue<T extends AWTEvent> =
    EventQueue>AWTEvent<;
```

`GEventQueue<T extends AWTEvent>` is a generic definition of `EventQueue` that contains a particular data type, `T`. A type has to be provided to `GEventQueue` in order to form a complete class definition. To satisfy the Java type checker, a constraint has to be set on `T` by enforcing it to be a subtype of `AWTEvent`. For example, in the method `postEvent(...)`, `getID()` is invoked on an event. The `getID()` method is defined in the class `AWTEvent`. Without the constraint on `T`, `GEventQueue<T>` would be rejected by the Java compiler since it can not statically guarantee the presence of `getID()`.

The generic `GEventQueue<T>` resulting from the snippet of code given above is equivalent to:

```
public class GEventQueue<T extends AWTEvent> {
    private synchronized T getCurrentEventImpl() {
        return (Thread.currentThread() == dispatchThread)
            ? ((T)currentEvent.get()): null;
    }
    public T getNextEvent() throws InterruptedException {
        ...
    }
    public void postEvent(T theEvent) {
        ...
        boolean notifyID = (theEvent.getID() == this.waitForID);
        ...
    }
    ...
}
```

References of `AWTEvent` have been replaced by the type parameter `T`. `GEventQueue` is free from references of the AWT event class definition. The queue may be employed with `KeyEvent` then, a subclass of `AWTEvent`:

```
GEventQueue<KeyEvent> keyEventsQueue
    = new GEventQueue<KeyEvent>();
keyEventsQueue.postEvent(new KeyEvent(...));
try {
    KeyEvent event = keyEventsQueue.getNextEvent();
} catch(Exception e) {} // getNextEvent() is throwable
```

Interface Generalization. The mechanism for classes described above may be applied to interfaces. For example, the AWT `ActionListener` interface is defined as follows:

```
public interface ActionListener extends EventListener {
    public void actionPerformed(ActionEvent e);
}
```

This interface may be generalized with the following declaration:

```
public interface GActionListener<T> =
    ActionListener>ActionEvent<;
```

The benefit of this generalization is the ability to reuse the interface `ActionListener` with a different event API.

Incremental Generalization. A generic class obtained using reverse generics may be generalized further by unbinding other remaining static type references. For instance, let us consider the class `EventDispatchThread`, which is a package-private AWT class which takes events off the `EventQueue` and dispatches them to the appropriate AWT components. `EventDispatchThread` is used in the `EventQueue` class as follows:

```
public class EventQueue {
    ...
    private EventDispatchThread dispatchThread;

    final void initDispatchThread() {
        synchronized (this) {
            if (dispatchThread == null &&
                !threadGroup.isDestroyed()) {
                dispatchThread = (EventDispatchThread)
                    AccessController.doPrivileged(new PrivilegedAction()
                    {...}})}
        }
    }
}
```

In the situation where some assumptions may have to be made on the type of the event dispatcher, the `GEventQueue<T extends AWTEvent>` may be generalized further:

```
public class
GenericEventQueue<Dispatcher extends EventDispatchThread>
    =GEventQueue>EventDispatchThread<;
```

The generic `GenericEventQueue` has two type parameters, `T` and `Dispatcher`. Note that the definition above is equivalent to the generic class obtained by unbinding both the `AWTEvent` and the `EventDispatchThread` type in one single step:

```
public class GenericEventQueue
    <T extends AWTEvent,
    Dispatcher extends EventDispatchThread>
    =EventQueue>AWTEvent, EventDispatchThread<;
```

The class `GenericEventQueue<T,Dispatcher>` can be instantiated by providing the proper two type parameters.

At the current stage of reverse generic, Incremental Generalization assumes that the two parameters are distinct types. If not, then the generalization cannot be applied.

Partial Instantiation. The generic `GenericEventQueue` described above may be partially instantiated by fulfilling only some of its type parameters. For example, an event queue dedicated to handle key events may be formulated:

```
public class GKeyEventQueue =
    GenericEventQueue<KeyEvent>;
```

One type argument has still to be provided to `GKeyEventQueue`, i.e., the one corresponding to the type parameter `Dispatcher`. A complete instantiation may be:

```
public class OptimizedKeyEventQueue
    = GKeyEventQueue<OptimizedEventDispatchThread>;
```

`OptimizedKeyEventQueue` has all the type parameters instantiated and can be used to create objects:

```
OptimizedKeyEventQueue keyEventsQueue
    = new OptimizedKeyEventQueue();
keyEventsQueue.postEvent(new KeyEvent(...));
try {
    KeyEvent event = keyEventsQueue.getNextEvent();
} catch (Exception e) {} // getNextEvent() is throwable
```

3 Dealing with the Language Features

Reverse generics is a general mechanism that can be used to extend an existing programming language that already provides a generic programming mechanism. However, since this mechanism relies on the existing generic programming features provided by the language under examination, we need to investigate whether such existing mechanisms are enough to create a complete “reversed generic” class. This section summarizes the various technical limitations of Java generics and their impact on reverse generics. In this respect, C++ does not seem to put limitations for reverse generics. A broader comparison between the generic programming mechanisms provided by Java and C++ may be found in the literature [11,6].

Class Instantiation. A crucial requirement in the design of the addition of generic in Java 1.5 was backward compatibility, and in particular to leave the Java execution model unchanged. *Erasure* [20,7] is the front-end that converts generic code into class definitions. It behaves as a source-to-source translation. Because of erasure, `List<Integer>` and `List<String>` are the same class. Only one class is generated when compiling the `List<T>` class. At runtime, those two instantiations of the same generic `List<T>` are just Lists. As a result, constructing variables whose type is identified by a generic type parameter is problematic.

Let’s consider the following code:

```
class PointFactory {
    Point create() { return new Point(); }
}
```

One might want to generalize `PointFactory` the following way:

```
class Factory<T> = PointFactory>Point<;
```

The corresponding reversed generics class would correspond to the following generic class:

```
class Factory<T> {
    T create() { return new T(); }
}
```

However, this class would not be well-typed in Java: a compile-time error is raised since `new Point()` cannot be translated into `new T()` for the reason given above. As a result, the class `PointFactory` cannot be made generic in Java. Enabling a generic type parameter to be instantiated is considered to be a hard problem [2]. A proposal has been made to eliminate such restrictions [1], in order to let generic types appear in any context where conventional types appear.

On the contrary, in C++, the above code for `Factory<T>` (with the corresponding C++ template syntax adjustments) is perfectly legal, and instantiation of generic types is also used in the STL itself.

Static Methods. Generic class type parameters cannot be used in a static method¹. A generic method has to be used instead.

For example, the class `EventQueue` contains the static method `eventToCacheIndex(...)`:

```
private static int eventToCacheIndex(AWTEvent e) {
    switch(e.getID()) {
        case PaintEvent.PAINT: return PAINT;
        case PaintEvent.UPDATE: return UPDATE;
        case MouseEvent.MOUSE_MOVED: return MOVE;
        case MouseEvent.MOUSE_DRAGGED: return DRAG;
        default: return e instanceof PeerEvent ? PEER : -1;
    }
}
```

This method is generalized in `GenericEventQueue` as the following:

```
private static <U extends AWTEvent>
    int eventToCacheIndex(U e) {
    ...
}
```

The type parameter `U` cannot be equal to `T` since `eventToCacheIndex(...)` is a static method. This means that `eventToCacheIndex(...)` may be employed with a type T_1 even if `GenericEventQueue` has been invoked with a type T_2 . For example, we might have:

¹ <http://java.sun.com/docs/books/tutorial/extra/generics/methods.html>

```

class GEventQueue<T extends AWTEvent>
    = EventQueue<AWTEvent>;...
GEventQueue<KeyEvent> keyEventQueue
    = new GEventQueue<KeyEvent>();
GEventQueue.eventToCacheIndex(new ActionEvent(...));

```

The expression `GEventQueue<KeyEvent>` instantiates the generic with the type `KeyEvent`. However, the static method `eventToCacheIndex` is performed with an `ActionEvent`, a subclass of `AWTEvent` living in a different class hierarchy than `KeyEvent`. While this does not undermine type safety, we believe that it might represent an abnormal situation with respect the initial design intentions. This issue suggests an extension of reverse generic to handle static methods as a possible further investigation.

On the contrary, C++ deals with generic class type parameters used in static methods, as in the following code (the implicit type constraint is that the operator `<<` is defined for `T`, which is the case for the basic types we use in `main`):

```

template<typename T>
class ClassWithStaticMethod {
    T myField;
public:
    // constructor initializing the field
    ClassWithStaticMethod(const T& t) : myField(t) {}

    static void m(T t) {
        cout << "t is: " << t << endl;
    }
};

int main() {
    ClassWithStaticMethod<int>::m(10);
    ClassWithStaticMethod<float>::m(10.20);
    ClassWithStaticMethod<string>::m("foobar");

    // create an object of class ClassWithStaticMethod<string>
    // passing a string argument
    ClassWithStaticMethod<string> c("value");
    c.m("hello");
    c.m(10); // compile ERROR!
}

```

Note also how the C++ compiler correctly detects the misuse of a static method in the last line: the static method of a class where the generic type is instantiated with `string` is being used with another type (`int`)².

Abstract Class. Turning a type contained in a method signature into a type parameter may make the resulting generic class abstract in Java. Consider the following two class definitions:

² A static method invoked on an instance corresponds to the static method invoked on the instance's class.

```

abstract class AbstractCell {
    public abstract void set (Object obj);
    public abstract Object get ();
}
class Cell extends AbstractCell {
    private Object object;
    public void set (Object obj) { this.object = obj; }
    public Object get () { return this.object; }
}

```

One may want to write the following generic to make `Cell` operate on `Number` instead of `Object`:

```
class GCell<T extends Number> = Cell>Object<;
```

However, `GCell` is abstract since `set(Object)` is not implemented. The solution is to make `AbstractCell` generic by abstracting `Object`. In C++, since it does not type check a generic class, but only its instantiations (*i.e.*, generated classes), the situation is different, as illustrated in the next section.

Method Erasure Uniqueness. Consider the previous code excerpt of `AbstractCell` and `Cell`. Let us assume one wants to make `Cell` operate on any arbitrary type instead of `Object`. Naively, one may write the following definition:

```
class GCell<T> = Cell>Object<;
```

It is the same definition of `GCell` provided above without the upper type. This definition results in a compile error. The reason is that the method `set` has two different erasures without being overriding. This is a further limitation of the Java type erasure.

To fully understand why, consider the following example. This code is rejected by the Java compiler (with the error “*GCell is not abstract and does not override abstract method set(java.lang.Object) in AbstractCell*”):

```

class GCell<T> extends AbstractCell {
    private T object;
    public void set (T obj) { this.object = obj; }
    public T get () { return this.object; }
}

```

This is due to the Java erasure mechanism which prevents two methods from having the same erasure if one does not override the other. The method `set(T)` in `GCell<T>` and `set(Object)` in `AbstractCell` have the same erasure. The former does not override the latter, but overloads it. An illustration of this limitation is:

```

class MyClass<U,V> {
    // These two overloaded methods are ambiguous
    void set (U x) { }
    void set (V x) { }
}

```

Defining an upper bound of the type `T` will enforce this overloading and removes the method erasure ambiguity. A compilable version could be:

```
class MyClass<U extends Object, V extends java.awt.Frame> {
    void set (U x) { }
    void set (V x) { }
}
```

Let us now consider a possible reversed generic `GCell` generated in C++, which could correspond to the following one:

```
class AbstractCell {
public:
    virtual void set(int o) = 0;
    virtual int get() = 0;
};

template<typename T>
class GCell: public AbstractCell {
    T object;
public:
    GCell(T o) : object(o) {}
    virtual void set(T o) { object = o; }
    virtual T get() { return object; }
};

int main() {
    AbstractCell *cell = new GCell<int> (10);
    cout << cell->get() << endl;
    cell->set(20);
    cout << cell->get() << endl;
    AbstractCell *cell2 =
        new GCell<string> ("foo"); // compile ERROR
}
```

C++ correctly considers `GCell<int>` as a concrete class since the abstract methods of the base class are defined³. However, if we tried to instantiate `GCell<string>` we would get a compiler error, since `GCell<string>` is considered abstract: in fact, the `get/set` methods in the abstract class are not implemented (`GCell<string>` defines the overloaded version with `string`, not with `int`).

Primitive Types. Arithmetic operators in Java have to be used in a direct presence of numerical types only. As an example, the `+` and `-` operators can only be used with primitive types and values. The autoboxing mechanism of Java makes it operate with the types `Integer`, `Float`.

For example, the following generic method is illegal since `Number` objects cannot be arguments of `+` and `-`:

³ To keep the example simple we used `int` as a type, since no `Object` is available in C++.

```
public class T<U extends Number> {
    public int sum (U x, U y) {
        return x + y;
    }
}
```

Instead, the following declaration of `sum` is legal:

```
public class T<U extends Integer> {
    public int sum (U x, U y) {
        return x + y;
    }
}
```

This means that one can reverse generic a class by abstracting the type `Integer` into a parameter `U extends Integer`. However, this would not be highly useful since `Integer` is a final class, `sum` can be applied only with the type `Integer`.

The use of arithmetic operations prevents the operand types from being turned into type parameters in a generic way. This is not a problem in C++ thanks to operator overloading (a feature that is still missing in Java), as also illustrated in the following section.

Operators. Java does not provide operator overloading, but it implements internally the overloading of `+`, for instance, for `Integer` and `String`. Thus, the two classes are legal in Java:

```
public class IntSum {
    public static Integer sum (Integer x, Integer y) {
        return x + y;
    }
}

public class StringSum {
    public static String sum (String x, String y) {
        return x + y;
    }
}
```

But there is no way to extract a generic version, since there is no way to write a correct type constraint⁴.

This is not a problem in C++ thanks to operator overloading. However, we think that this problem is not strictly related to the absence of operator overloading in Java. Again, It is due to type erasure and how the type-checking is performed in Java. C++ does not perform type-checking on the generic class: upon type parameter instantiation it type-checks the resulting (implicitly) instantiated class; thus, we can write in C++ such a generic class with method `sum`, which will have only some accepted (well-typed) instantiations, i.e., those that satisfy the implicitly inferred constraints (in our

⁴ This might be solved, possibly, with a *union type* [12] constraint such as, e.g., `extends Integer ∨ String`.

case, the operator + must be defined on the instantiated type). On the contrary, Java type-checks the generic class itself, using the explicit constraint, which in our case, cannot be expressed in such a way that it is generic enough.

4 Discussion

In this section we discuss how the reverse generics introduced features are related to the underlying programming language (in our case study, Java and C++). Moreover, we hint some possible usage of reverse generics in a development scenario.

Partial Template Specialization. C++ allows the programmer to write template specializations, i.e., special implementations of possibly partially instantiated generic classes and functions. This mechanism, besides being the base for *template metaprogramming* [23,4,3], is very useful for providing optimized implementations for specific types; e.g., if there is a generic class storing data in a parameterized vector, we could write a specialized version for boolean using a single bit for each element and bitmasking, as sketched in the following example:

```
template <typename T>
class MyVector<T> {
    T *buffer; // initialized dynamically
public:
    // constructor omitted
    void set(int i, T t) { buffer[i] = t; }
    T get(int i) { return buffer[i]; }
}

// optimized version for bool
template <>
class MyVector<bool> {
    bool *buffer; // initialized dynamically
public:
    // constructor omitted
    void set(int i, bool t) { /* use bit masking */ }
    bool get(int i) { /* use bit masking */ }
}
```

Java, instead, does not provide this functionality. One might think of using incremental generalization and partial instantiation mechanisms of reverse generics (Sections 2 and 2, respectively). However, this is not enough to implement a mechanism corresponding to partial template specialization in Java. In fact, in C++, as shown above, we provide a specialized version for a type instance of a generic class, using the same class name (note that there is no inheritance involved), and of course this is impossible in Java due to the type erasure mechanism. Moreover, incremental generalization and partial instantiation, in reverse generics do not aim to mimic partial template specialization: it is a mechanism to create new generic classes from existing classes, by parameterizing (respectively, instantiating) only some types.

Development Methodology. Since highly parameterized software is harder to understand [10], we may think of a programming methodology where a specific class is developed and tested in a non-generic way, and then it is available to the users via its “reversed” generic version (thus, in this case, we really need the non generic version for testing purposes, so the code must not be refactored). For example, C++ debuggers may have problems when setting a breakpoint for debug purposes within a template from a source file: they may either miss setting the breakpoint in the actual instantiation desired or may set a breakpoint in every place the template is instantiated. Another well-known problem with programming using templates is that usually the C++ compilers issue quite long compilation errors in case of problems with template usage and in particular with template instantiations; these errors are also hard to understand due to the presence of parametric types.

Thus, reverse generics can be used as a development methodology, not only as a way to turn previous classes into generic: one can develop, debug and test a class with all the types instantiated, and then expose to the “external world” the generic version created through reverse generics. Provided that an explicit dependency among reversed generic classes and the original ones is assumed (e.g., by using makefiles), the reversed generic version of a class will be automatically kept in sync with the original one. This also shows that reverse generic incremental generalization and partial instantiation mechanisms are completely different mechanisms with respect to C++ template partial specialization: in the C++ code snippet above, it is up to the programmer to manually keep in sync the generic version of a class and all its possible template specializations.

Class Relations. Classes obtained with reverse generics are not related to the original classes. Thus, using the following instruction

```
class MyGenericClass<T extends MyType> =
    MyClass>MyType<;
```

`MyGenericClass` and `MyClass` are two distinguished and unrelated classes: the only thing that couples them (implicitly) is the fact that `MyGenericClass` has exactly the same code as `MyClass` modulo the type that was parameterized.

We think that this is the only sensible design choice since generic types and inheritance are basically two distinguished features that should not be mixed; indeed the main design choices of Java generics tend to couple generics and class based inheritance (again, for backward compatibility), relying on type erasure, and, as we discussed throughout the paper, this highly limits the expressivity and usability of generics in a generic programming methodology.

C++ keeps the two above features unrelated; in particular, the STL library basically does not rely on inheritance at all [17,18,5], leading to a real usable generic library (not to mention that, avoiding inheritance and virtual methods also leads to an optimized performance).

Conservative Extension. Reverse generics is a conservative linguistic extension that preserves the original type system and semantic rules of the underlying programming language. Thus, the features introduced by our linguistic extension are independent from the hosting language. It is important to notice that the resulting generic classes might not be typable in the extended programming language: type erasure plays an

essential role (Section 3). The main idea is that the code implicitly produced by a reversed generic class corresponds to what a programmer might have written manually, and should be typable by the extended language accordingly.

For instance, the following instruction

```
class MyClass<TypeName> = MyClass>TypeName<;
```

is legal from the reverse generics mechanism's point of view. However, in Java there is no way to write a typable corresponding code (due to type erasure we would end up having two classes with the same name).

In C++ the original class could be seen as a specialization of the reversed generic one. However, the following possible corresponding code is rejected, due to the unparameterized class:

```
template<typename T>
class MyClass {
    T myField;
public:
    void set(T f) { myField = f; }
    T get() { return myField; }
};
```

```
class MyClass {
    int myField;
public:
    void set(int f) { myField = f; }
    int get() { return myField; }
};
```

Since in the presence of template specialization, the template declarations must be explicit also in the specialized class, thus, we should change the second class as follows:

```
template <>
class MyClass<int> {
    int myField;
public:
    void set(int f) { myField = f; }
    int get() { return myField; }
};
```

A possible solution could be to change also the original code accordingly during the reverse generic operation. However, this would make the approach less language independent.

5 Related Work

To our knowledge, no programming language construct to build a generic class from a complete class definition has been presented in the literature. This section presents the closest work to Reverse Generics.

Reverse Engineering Parameterized Types. A first attempt to automatically extract generic class definitions from an existing library has been conveyed by Duggan [9], well before the introduction of generics into Java.

Beside the reverse engineering aspect, Duggan's work diverges from Reverse Generics regarding downcast insertion and parameter instantiation. Duggan makes use of *dynamic subtype constraint* that inserts runtime downcast. Parameterized type may be instantiated, which requires some type-checking rules for the creation of an object: the actual type arguments must satisfy the upper bounds to the formal type parameters in the class type. Moreover, the version of generics presented in his work with Poly-Java differs from Java 1.5 in several important ways that prevent his results from being applied to Java generics.

Modular Type-based Reverse Engineering. Kiezun et al. proposes a type-constraints-based algorithm for converting non-generic libraries to add type parameters [14]. It handles the full Java language and preserves backward compatibility. It is capable of inferring wildcard types and introducing type parameters for mutually-dependent classes.

Reverse engineering approaches ensure that a library conversion preserves the original behavior of the legacy code. This is a natural intent since such a conversion is exploited as a refactoring. The purpose of Reverse Generics is to replace static types references contained in existing classes with specialized ones. Section 3 shows that a generic obtained from a complete class may have to be set abstract. This illustrates that the original behavior of the complete class may not be preserved in the generic ones. Method signatures may be differently resolved in the generic class.

Type Construction Polymorphism. A well-known limitation of generic programming in mainstream languages is to not be able to abstract over a type constructor. For instance, in `List<T>`, `List` is a type constructor, since, given an argument for `T`, e.g., `Integer`, it builds a new type, i.e., `List<Integer>`. However, the type constructor `List` itself cannot be abstracted (this is a well known limitation of first-order parametric polymorphism). Thus, one cannot pass a type constructor as a type argument to another type constructor. Moors, Piessens and Odersky [16] extend the Scala language [19] with type construction polymorphism to allow type constructors as type parameters. Thus, it is possible not only to abstract over a type, but also over a type constructor; for instance, a class can be parameterized over `Container[T]`⁵, where `Container` is a type constructor which is itself abstracted and can be instantiated with the actual collection, e.g., `List` or `Stack`, which are type constructors themselves.

Reverse generics act at the same level of first-order parametric polymorphism, thus, it shares the same limitations, e.g., the following reverse generic operation cannot be performed:

```
class MyClass {
    List<Integer> mylist;
}

class MyClassG = MyClass>List<;
```

⁵ Scala uses `[]` instead of `<>`.

An interesting extension is to switch to the higher level of type constructor polymorphism, but this is an issue that still needs to be investigated, and, most important, it should be experimented with a programming language that provides type constructor polymorphism, and, with this respect, Scala seems the only choice compared to Java and C++.

6 Conclusions

Genericity in programming languages appeared in the beginning of the 70s. It gained a large adoption by being adopted in mainstream languages. All the generic mechanisms we are aware of enable a parameterization only if the code has been prepared for being parametrized. This paper goes against this implicitly established mindset. Reverse generics promote a generalization for code that has not been prepared for it.

Since highly parameterized software is harder to understand [10], we may think of a programming methodology where a specific class is developed and tested in a non-generic way, and then it is available to the users via its “reversed” generic version (thus, in this case, we really need the non generic version for testing purposes, so the code must not be refactored). For example, C++ debuggers may have problems when setting a breakpoint for debug purposes within a template from a source file: they may either miss setting the breakpoint in the actual instantiation desired or may set a breakpoint in every place the template is instantiated. Another well-known problem with programming using templates is that usually the C++ compilers issue quite long compilation errors in case of problems with template usage and in particular with template instantiations; these errors are also hard to understand due to the presence of parametric types.

Thus, reverse generics can be used as a development methodology, not only as a way to turn previous classes into generic: one can develop, debug and test a class with all the types instantiated, and then expose to the “external world” the generic version created through reverse generics. Provided that an explicit dependency among reversed generic classes and the original ones is assumed (e.g., by using makefiles), the reversed generic version of a class will be automatically kept in sync with the original one.

Classes obtained with reverse generics are not related to the original classes. We think that this is the only sensible design choice since generic types and inheritance are basically two distinguished features that should not be mixed; indeed the main design choices of Java generics tend to couple generics and class based inheritance (again, for backward compatibility), relying on type erasure, and, as we discussed throughout the paper, this highly limits the expressivity and usability of generics in a generic programming methodology. C++ keeps the two above features unrelated; in particular, the STL library basically does not rely on inheritance at all [17,18,5], leading to a real usable generic library (not to mention that, avoiding inheritance and virtual methods also leads to an optimized performance).

We currently described reverse generics in a very informal way by describing a surface syntax and its application to Java and C++. We plan to investigate the applicability of reverse generics also to other programming languages with generic programming capabilities such as, e.g., C# and Eiffel [15].

As a future work, we will seek a stronger and deeper theoretical foundation. The starting point could be Featherweight Java [13], a calculus for a subset of Java which

was also used for the formalization of Java generics. Alternatively, we might use the framework of [21], which, working on C++ templates that provide many more features than Java generics, as we saw throughout the paper, seem to be a better candidate for studying the advanced features of reverse generics.

References

1. Allen, E., Bannet, J., Cartwright, R.: A First-Class Approach to Genericity. In: Proc. of OOPSLA, pp. 96–114. ACM, New York (2003)
2. Allen, E.E., Cartwright, R.: Safe instantiation in generic java. *Sci. Comput. Program.* 59(1-2), 26–37 (2006)
3. Abrahams, D., Gurtovoy, A.: C++ Template Metaprogramming: Concepts, Tools and Techniques from Boost and Beyond. Addison-Wesley, Reading (2004)
4. Alexandrescu, A.: Modern C++ Design, Generic Programming and Design Patterns Applied. Addison Wesley, Reading (2001)
5. Austern, M.H.: Generic Programming and the STL: using and extending the C++ Standard Template Library. Addison-Wesley, Reading (1998)
6. Batov, V.: Java generics and C++ templates. *C/C++ Users Journal* 22(7), 16–21 (2004)
7. Bracha, G., Odersky, M., Stoutamire, D., Wadler, P.: Making the future safe for the past: adding genericity to the Java programming language. In: Proc. of OOPSLA, pp. 183–200. ACM, New York (1998)
8. Reis, G.D., Järvi, J.: What is generic programming? In: Proc. of LCSD (2005)
9. Duggan, D.: Modular type-based reverse engineering of parameterized types in java code. In: Proc. of OOPSLA, pp. 97–113. ACM, New York (1999)
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Reading (1995)
11. Ghosh, D.: Generics in Java and C++: a comparative model. *ACMSIGPLAN Notices* 39(5), 40–47 (2004)
12. Igarashi, A., Nagira, H.: Union Types for Object Oriented Programming. *Journal of Object Technology* 6(2), 31–52 (2007)
13. Igarashi, A., Pierce, B.C., Wadler, P.: Featherweight Java: a minimal core calculus for Java and GJ. *ACM TOPLAS* 23(3), 396–450 (2001)
14. Kiezun, A., Ernst, M.D., Tip, F., Fuhrer, R.M.: Refactoring for parameterizing java classes. In: Proc. of ICSE, pp. 437–446. IEEE, Los Alamitos (2007)
15. Meyer, B.: Eiffel: The Language. Prentice-Hall, Englewood Cliffs (1992)
16. Moors, A., Piessens, F., Odersky, M.: Generics of a higher kind. In: Proc. of OOPSLA, pp. 423–438. ACM, New York (2008)
17. Musser, D.R., Stepanov, A.A.: Generic programming. In: Gianni, P. (ed.) ISSAC 1988. LNCS, vol. 358, pp. 13–25. Springer, Heidelberg (1989)
18. Musser, D.R., Saini, A.: STL Tutorial and Reference Guide. Addison Wesley, Reading (1996)
19. Odersky, M., Spoon, L., Venners, B.: Programming in Scala. Artima (2008)
20. Odersky, M., Wadler, P.: Pizza into Java: Translating theory into practice. In: Proc. of POPL, pp. 146–159. ACM, New York (1997)
21. Siek, J., Taha, W.: A semantic analysis of C++ templates. In: Hu, Q. (ed.) ECOOP 2006. LNCS, vol. 4067, pp. 304–327. Springer, Heidelberg (2006)
22. von Dinkelage, D., Diwan, A.: Converting Java classes to use generics. In: Proc. of OOPSLA, pp. 1–14. ACM, New York (2004)
23. Veldhuizen, T.: Using C++ template metaprograms. *C++ Report* 7(4), 36–43 (1995)

A Calculus of Agents and Artifacts^{*}

Ferruccio Damiani¹, Paola Giannini², Alessandro Ricci³, and Mirko Viroli³

¹ Dipartimento di Informatica, Università di Torino, Torino, Italy

² Dipartimento di Informatica, Università del Piemonte Orientale, Piemonte, Italy

³ DEIS, Alma Mater Studiorum, Università di Bologna, Bologna, Italy

Abstract. A library-based extension of JAVA, the SIMPA framework, introduced a new abstraction based on agent-oriented concepts. *Agents* are autonomous entities that cooperate by exploiting *artifacts*, representing resources that are dynamically created and shared by agents. In this paper we present a core calculus integrating techniques coming from the area of concurrency and from OO programming. The syntax of the calculus with its static and dynamic semantics are introduced through an example. The calculus aims to foster the formalization (and proof) of type soundness of SIMPA programs and the development of techniques for analyzing the computational behaviour of agents and artifacts.

1 Introduction

Multi-core architectures, Internet-based computing and Service-Oriented Architectures/ Web Services, are increasingly introducing concurrency issues (and distribution) in the context of a large class of applications and systems—up to making them key factors of almost *any* complex software system. As noted in [15], even though concurrency has been studied for about 30 years in the context of computer science fields such as programming languages and software engineering, this research has not significantly impacted on mainstream software development. However, it appears more and more important to introduce higher-level abstractions, which can “help build concurrent programs, just as object-oriented abstractions help build large component-based programs” [15].

The A&A (Agents and Artifacts) meta-model, recently introduced in the context of agent-oriented programming and software engineering as a novel foundational approach for modelling and engineering complex software systems [10], goes in this direction. *Agents* and *artifacts* are the basic high-level and coarse-grained abstractions available in A&A: agents are used to model (pro)-active and task-oriented components of a system, which encapsulate the logic and control of their execution, while artifacts model purely-reactive function-oriented components of a system, used by agents to support their (individual and collective) activities.

In [12,14] it is introduced SIMPA, a library-based extension of JAVA providing programmers with *agent-oriented abstractions* on top of the basic OO layer, to be used as basic building blocks to define the architecture of complex (concurrent) applications. In

^{*} Work partially supported by MIUR PRIN 2009 DISCO project. The funding bodies are not responsible for any use that might be made of the results presented here.

SIMPA, the underlying OO computational model of JAVA is still adopted, but only for defining agents and artifacts programming and data storage, namely, for defining the purely computational part of applications. On the other hand, agents and artifacts are used to define aspects related to system architecture, interaction, and synchronisation.

In this paper we promote the applicability of A&A metamodel in OO programming a step further, by introducing FAL (FEATHERWEIGHT AGENT LANGUAGE), a core calculus formalizing the key features of SIMPA. The formalization is largely inspired to FJ, (FEATHERWEIGHT JAVA) [7], and is based on reduction rules applied at certain evaluation contexts. On the other hand, being concurrency-oriented, this calculus uses techniques coming from concurrency theory, as e.g. in process algebras. A system configuration is seen as a parallel composition of agents and artifacts instances (seen as independent and asynchronous processes), the former keeping track of a tree of (sub-)activities to be executed in autonomy, the latter holding a set of pending operations to be executed in response to agent actions over the artifact.

The remainder of the paper is organised as follows. Section 2 introduces the SIMPA programming model. Section 3 presents syntax, operational semantics of the FAL calculus, and discusses briefly the properties that follow from type soundness. The operational semantics is presented via an example. For lack of space the formal rules are not given. Section 4 discusses some related work and concludes by outlining possible directions for further work.

2 The Programming Model

In this section we describe an abstract version of SIMPA programming model by exploiting the syntax of the FAL calculus.

The Agent Programming Model. In essence, an agent in SIMPA is a stateful entity whose job is to pro-actively execute a structured set of *activities* as specified by the agent programmer, including possibly non-terminating activities, which finally result in executing sequences of *actions*, either internal actions – inspecting/changing its own state – or external actions – interacting with its environment. All actions are executed atomically.

The state of an agent is represented by an associative store, called *memo-space*, which represents the long-term memory where the agent can dynamically attach, associatively read and retrieve chunks of information called *memo*. A memo is a tuple, characterised by a label and an ordered set of arguments, either bound or not to some data object (if some is not bound, the memo is hence partially specified). For instance, the philosopher agent uses a memo *hungry* to take note that its state is now *hungry* and it needs the forks, and *stopped* to keep track that it needs to terminate. A basic set of internal actions is available to agents to work atomically with the memo-space: `+memo` is used to create a new memo with a specific label and a variable number of arguments, `?memo` and `-memo` to *get/remove* a memo with the specified label.

The computational behaviour of an agent can be defined as a hierarchy of activities (corresponding to the execution of some tasks). Activities can be simple or structured. A simple activity is composed by just a flat sequence of actions, as a single control flow,

```

agent Main {
  activity main() { Table t = make Table(new boolean[5]);
    spawn Philosopher(0,1,t); spawn Philosopher(1,2,t);
    spawn Philosopher(2,3,t); spawn Philosopher(3,4,t);
    spawn Philosopher(4,0,t); }
}
artifact Table { boolean[] isBusyFork;
  operation getForks(int left, int right)
  :guard ((not(.isBusyFork[left]) and (not(.isBusyFork[right])))
  { .isBusyFork[left] = true; .isBusyFork[right] = true;
    signal(forks_acquired);
  }
  operation releaseForks(int left, int right) :guard true
  { .isBusyFork[left] = false; .isBusyFork[right] = false; }
}
agent Philosopher { Sns s;
  activity main(int left, int right, Table table)
  :agenda ( prepare() :pre true,
            living(left,right,table) :pre memo(hungry)
            :pers not(memo(stopped)) ) { }
  activity prepare() { +memo hungry; }
  activity living(int left, int right, Table table)
  :agenda ( eating(left,right,table) :pre memo(hungry),
            thinking() :pre completed(eating),
            shutdown() :pre failed(eating) ) { }
  activity thinking() { ... /* think */ +memo hungry; }
  activity eating(int left, int right, Table table)
  { use table.getForks(left,right) :sns s
    sense s :filter forks_acquired;
    ... /* eat */
    use table.releaseForks(left,right);
    -memo(hungry);
  }
  activity shutdown() { +memo(stopped); }
}

```

Fig. 1. The five dining philosophers problem

while structured activities have a non-empty agenda specifying sub-activities, which in turn can be possibly executed in the context of such super-activity—hence leading to the hierarchical structure of behaviour. At the language level, simple activities are represented by `activity` blocks, providing the name of the activity and parameters. By default each agent has a `main` activity, which can be either simple or structured. In the dining philosophers example shown in Figure 1, the `Philosopher` agent has the simple activities, `prepare`, `eating`, `thinking`, and `shutdown`. A structured activity has a non-empty *agenda*, specifying a set of *todos* representing sub-activities that must be executed in the context of the parent activity—also called super-activity. In the philosophers example, `main` and `living` are structured activities. A *todo* contains the name of the sub-activity to be executed, a precondition over the inner state of the

agent that must hold for the specified sub-activity to start, and attributes related to sub-activity execution, such as persistency. Preconditions are expressed as a boolean expression over a basic set of predefined predicates. Essentially, the predicates make it possible to specify conditions on the current state of the activity agenda, in particular on (i) the state of the sub-activities (if they started, completed, or aborted) and on (ii) the local inner state of the agent, that is the memo space. For instance, the predicate `memo (M)` is true if the specified memo `M` is found in the memo space. In the example, in the structured activity `living`, sub-activity `eating` is executed as soon as a memo `hungry` is found in the memo space. When the precondition of a `todo` item holds (for an activity in execution listing such `todo` in the agenda), the `todo` is removed from the agenda and an instance of the sub-activity is created and executed. So, multiple sub-activities can be executed concurrently and asynchronously, in the context of the same parent activity. Sub-activities execution can be then synchronized by properly specifying preconditions in `todos`, hence in a declarative way. If a `todo` is declared persistent, as soon as the sub-activity is completed the `todo` is re-inserted into the agenda. The persistency attribute can specify also the condition under which the activity should persist. For instance, the `todo` item about `living` sub-activity in `philosopher` agent is declared persistent until a `stopped` memo is found.

The Artifact Programming Model. An artifact is composed by three main parts: (i) observable properties, which are attributes that can be observed by agents without an explicit agent action towards the artifact; (ii) a description of the inner non-observable state, composed by set of state variables analogous to private instance fields of objects; and (iii) *operations*, which embody the computational behaviour of artifacts. The `Table` artifact in the `philosopher` example in Figure 1 has no observable properties, an inner state variable `isBusyFork`, an array of booleans, and two operations, `getForks` and `releaseForks`, the first used to acquire two forks and the latter for releasing forks. Both state variables and observable properties are declared similarly to instance fields in objects; observable properties are prefixed by `obsprop` qualifier. In both cases, a dot notation (e.g. `.isBusyFork`) is used both for l-value and r-value, to syntacatically distinguish them from parameters.

Operations can be defined by method-like blocks qualified as `operation`, specifying the name and parameters of the operation and a computational body. It is worth noting that no return parameter is specified, since operations in artifacts are not exactly like methods in objects. For each `operation`, implicitly an *interface control* in the usage interface is defined, with the specified signature. Operations can be either *atomic*, executed as a single computational step, or *structured*, i.e. composed by multiple atomic operation steps. For sake of space, in this paper we consider only atomic operations. For each operation a *guard* can be specified (`:guard` declaration), representing the condition that must hold for the related control in the usage interface to be enabled. For instance, the `getForks` operation in `Table` artifact is available – i.e. the related control is enabled in the usage interface – when the specified forks are not busy.

To be useful, an artifact typically should provide some level of *observability*. This is achieved both by generating observable events through the `signal` primitive, and by defining observable properties. In the former case, the primitive generates observable


```

artifact Counter { obsprop int count;
  Counter(int c){ .count = c; }
  operation inc() { .count = .count+1; }
}
agent Main {
  activity main() {
    Counter c = make Counter(0);
    spawn Observer(c); spawn User(c); spawn User(c); }
}
agent Observer { Sns s;
  activity main(Counter c)
    :agenda ( prepare(c),
              monitoring(c) :pre completed(prepare)
              :pers (not memo(finished)) { }
  activity prepare(Counter c) { focus (c,s); }
  activity monitoring(Counter c) {
    sense s :filter prop_updated;
    int value = observe c.count;
    ... // do something
    if (value >= 100 ){ +memo(finished); } }
}
agent User {
  activity main(Counter c)
    :agenda ( usingCount(c) :pers true ) {}
  activity usingCount(Counter c) { use c.inc(); }
}

```

Fig. 2. A simple program with an Observer agent continuously observing a Counter Artifact, which is concurrently used by two User agents

events that can be observed by the agent using the artifact – i.e. by the agent which has executed the operation. An observable event is represented by a labelled tuple, whose label represents the kind of the event and the information content. For instance, in the Table artifact `getForks` operation generates the `forks_acquired(Left, Right)` tuple. Actually, the observable event `op_exec_completed` is automatically generated – without explicit signals – as soon as the execution of an operation is completed. In the latter case, observable properties are instance variables qualified as `obsprop`. Any time the property changes, an observable event of type `prop_updated` is fired with the new value of the property as a content. The observable events is observed by all the agents that are *focussing* (observing) the artifact (more details in next subsection). An example of simple artifact with observable properties is the Counter artifact shown in Figure 2: this artifact – working as an observable counter – has just a single observable property named `count` and an `inc` operation to update this count. Each time the operation is executed, the observable property and the event `prop_update(count, Value)` are automatically generated.

The Agent-Artifact Interaction Model. As already stated, artifact *use* and *observation* are the basic form of interaction between agents and artifacts. Artifact use by an agent

involves two basic aspects: (i) executing operations on the artifact, and (ii) perceiving through agent *sensors* the observable events generated by the artifact. Conceptually sensors represent a kind of “perceptual memory” of the agent, used to detect events coming from the environment, organize them according to some policy – e.g. FIFO and priority-based – and finally make them available to the agent. In the abstract language presented here, sensors used by an agent are declared at the beginning of the agent block.

In order to trigger operation execution, the *use* action is provided, specifying the target artifact, the operation to execute – or, more precisely, the usage interface control to act upon, which activates the operation – and optionally, a timeout and the identifier of the sensor used to collect observable events generated by the artifact. The action is blocked until either the action execution succeeds – which means that the specified interface control has been finally selected and the related operation has been started – or fails, either because the specified usage interface control is invalid (for instance it is not part of the usage interface) or the timeout occurred. If the action execution fails an exception is generated. In the philosopher example, a *Philosopher* agent (within its *eating* activity) executes a *use* action so as to execute the *getForks* operation, specifying the *s* sensor. On the artifact side, if the forks are busy the *getForks* usage interface control is not enabled, and the *use* is suspended. As soon as the forks become available the operation is executed and the *use* action succeeds.

It is important to note that no control coupling exists between an agent and an artifact while an operation is executed. However, operation triggering is a synchronization point between the agent (user) and the artifact (used): if the *use* action is successfully executed, then this means that the execution of the operation on the artifact has started.

In order to retrieve events collected by a sensor, the *sense* primitive is provided. The primitive waits until either an event is collected by the sensor, matching the pattern optionally specified as a parameter (for data-driven sensing), or a timeout is reached, optionally specified as a further parameter. As result of a successful execution of a *sense*, the event is removed from the sensor and a perception related to that event is returned. In the philosopher example, after executing *getForks* the philosopher agent blocks until a *forks_acquired* event is perceived on the sensor *s*. If no perception are sensed for the duration of time specified, the action generates an exception. Pattern-matching can be tuned by specifying custom event-selection filter: the default filter is based on regular-expression patterns, matched over the event type (a string).

Besides sensing events generated when explicitly using an artifact, a support for *continuous observation* is provided. If an agent is interested in observing every event generated by an artifacts – including those generated as a result of the interaction with other agents – two actions can be used, *focus* and *unfocus*. The former is used to start observing the artifact, specifying a sensor to be used to collect the events and optionally the filter to define the set of events to observe. The latter one is used to stop observing the artifact. In the example shown in Figure 2, an *Observer* agent continuously observes a *Counter* artifact, which is used by two *User* agents. After executing a *focus* on the artifact in the *prepare* activity, in the *monitoring* activity the observer prints on a console artifact the value of the observable property *count* as soon as it changes.

$U ::= G \mid A \mid C$	Agent / artifact / basic value types
$T ::= U \mid \text{Sns}$	Types
$GD ::= \text{agent } G \{ \text{Sns } \bar{s}; \overline{ACT} \}$	Agent (class) definition
$Act ::= \text{activity } a(\overline{T} \bar{x}) : \text{agenda}(\overline{\text{SubAct}}) \{e; \}$	Activity definition
$\text{SubAct} ::= a(\bar{e}) : \text{pers } e : \text{pre } e$	Subactivity definition
$AD ::= \text{artifact } A \{ \overline{U} \bar{x}; \overline{U} \bar{p}; \overline{Op} \}$	Artifact (class) definition
$Op ::= \text{operation } o(\overline{U} \bar{x}) : \text{guard } e \{e; \}$	Operation definition
$e ::= x \mid c$	Expressions: variable / basic value
$\mid \text{spawn } G(\bar{e}) \mid \text{make } A(\bar{e})$	agent and artifact instance creation
$\mid e; e$	sequential composition
$\mid .f \mid .f = e$	artifact-field access / update
$\mid .p \mid .p = e$	artifact-property access / update
$\mid \text{signal}(l(e))$	event generation
$\mid .s \mid \text{use } e.o(\bar{e}) : \text{sns } e \mid \text{sense } e : \text{filter } l$	sensor / operation use / event sensing
$\mid \text{focus}(e, e) \mid \text{unfocus}(e, e)$	focus / unfocus
$\mid \text{observe } e.p$	get property value
$\mid ?\text{memo}(l) \mid -\text{memo}(l) \mid +\text{memo}(l(e))$	memo operations
$\mid \text{memo}(l)$	memo predicate
$\mid \text{started}(a) \mid \text{completed}(a) \mid \text{failed}(a)$	activity state predicates
$\mid \text{fail}$	activity error

Fig. 3. Syntax

3 The Core Calculus

Syntax. The syntax of FAL is summarised in Figure 3 where we assume a set of basic values, ranged over by the metavariable c . Types for basic values are ranged over by the metavariable C . We only assume the basic values `true` and `false` (of type `Bool`) which are used as the result of the evaluation of preconditions, persistency predicates and guards. We use the overbar sequence notation according to [7].

There are minor differences between the syntax of the calculus and the one of the language used for the examples. Namely: instead of tuples for memos in memo-spaces (and event in sensors) we use values; and specifiers (`:agenda`, `:pers`, `:pre`, `:guard` and `:sns`), that are optional in the language, are mandatory in the calculus.

Labels are used as keys for the associative maps representing the content of sensors and memo-spaces. The metavariable l range over labels.

The expression `fail` model failures in activities, such as the evaluation of `?memo(l)` and `-memo(l)` in an agent in which the memo-space does not have a memo with label l . Note that the types of parameters, in artifact operations and the type of fields and properties may not be sensors so artifacts. Moreover, the signal expression, `signal(l(e))`, does not specify a sensor. Therefore, sensors may not be explicitly manipulated by artifacts.

The language is provided with a standard type system enforcing the fact that expressions occur in the right context (artifact or agent), operation used, and activities mentioned in todo lists are defined, and only defined fields and properties are accessed/modified.

Operational Semantics. The operational semantics is described by means of a set of reduction rules that transform sets of instances of agents/artifacts/sensors (configurations). In this section we first define configurations, and then introduce the reduction rules, via examples.

Configurations are non-empty sets of agent/artifact/sensor instances. Each agent/artifact/sensor instance has a unique identity, provided by a *reference*. The metavariable γ ranges over references to instance of agents, α over artifacts, σ over sensors.

Sensor instances are represented by $\sigma = \langle \bar{1} \bar{v} \rangle^{\text{Sns}}$, where σ is the instance identifier, and $\bar{1} \bar{v}$ is the queue of association labels/values representing the events generated (and not yet perceived) on the sensor.

Agent instances are represented by $\gamma = \langle \bar{1} \bar{v}, \bar{\sigma}, R \rangle^{\text{G}}$, where γ is the agent identifier, G is the type of the agent, $\bar{1} \bar{v}$ is the content of the *memo-space*, $\bar{\sigma}$ is the sequence of references to the instances of the sensors that the agent uses to perceive, and R is the state of the activity, *main*, that was started when the agent was created. The sensor instances in $\bar{\sigma}$ are in one-to-one correspondence with the sensor variables declared in the agent and are needed since every agent uses its own set of sensor instances.

An *instance of an activity*, R, describes a running activity. As explained in Section 2, before evaluating the body of an activity we have to complete the execution of its sub-activities, so we also represent the state of execution of the sub-activities.

$$R ::= a(\bar{v})[Sr_1 \cdots Sr_n]\{e\} \mid \text{failed}^a$$

The name of the activity is a , \bar{v} are the actual parameters of the current activity instance, $Sr_1 \cdots Sr_n$ is the set of sub-activities running, and e is the state of evaluation of the body of the activity. (Note that the evaluation of the body starts only when all the sub-activities have been fully evaluated.) With failed^a we say that activity a has *failed*. If the evaluation of a sub-activity is successful then it is removed from the set $Sr_1 \cdots Sr_n$. So when $n = 0$ starts the evaluation of the body e .

For a sub-activity, Sr , the process of evaluating its precondition (we do not consider the persistency predicate that would be similar), is represented by the term, $a(\bar{v})\langle e \rangle$ where e is different from `true` or `false` (it is the state of evaluation of the precondition) when $e = \text{true}$, the term $a(\bar{v})\langle \text{true} \rangle$ is replaced with the initial state of the evaluation of the activity a with parameters \bar{v} . When $e = \text{false}$ the evaluation of the precondition of a is rescheduled. Therefore:

$$Sr ::= a(\bar{v})\langle e \rangle \mid R$$

Artifact instances are represented by $\alpha = \langle \bar{f} = \bar{v}, \bar{p} = \bar{w}, \bar{\sigma}, O_1 \cdots O_n \rangle^A$ where α is the artifact identifier, A the type of the artifact, the sequence of pairs $\bar{f} \bar{v}$ associates a value to each the field of A, the sequence of pairs $\bar{p} \bar{w}$ associates a value to each property of A, the sequence $\bar{\sigma}$ represents the sensors that agents focusing on A are using, and O_i , $1 \leq i \leq n$, are the operations that are in execution. We consider $O_1 \cdots O_n$ a queue with first element O_n and last O_1 . (For simplicity, we do not consider steps in this paper, although we have a full formalization including them.) Artifacts are single threaded and (differently from agents that may have more activity running at the same time) only the operation O_n is being evaluated.

A *running operation*, O, is defined as follows.

$$O ::= (\sigma, o\langle e \rangle\{e'\})$$

where σ identifies the sensor associated with the operation which was specified by the agent containing the use that started the operation, and that is used to collect events

generated during the execution of the operation by `signal`. If the expression $\langle e \rangle$ is different from `true` or `false` the operation is evaluating its guard e . If $e = \text{true}$ then the operation is evaluating its body. If $e = \text{false}$ then the operation is removed from the queue and put at the end of it so that when it will be rescheduled it will restart evaluating its guard.

The *initial configuration* for the program in Fig. 1 is:¹

$$(1) \ \gamma_{\text{Main}} = \langle \emptyset, \emptyset, \text{main} \left[\begin{array}{l} \boxed{\text{Tablet} = \text{make Table}(\text{new Bool}[5])}; \\ \text{spawn Philosopher}(0, 1, \tau); \\ \dots \\ \text{spawn Philosopher}(4, 0, \tau) \end{array} \right] \rangle \{ \}^{\text{Main}}$$

This means that at the beginning we have only the agent `Main` with the running activity `main`, that does not have any subactivity, so the execution of its body starts from the first instruction. In the following we show how the *reduction rules* transform this initial term. The semantics is non deterministic. At each step,

- either an expression in an activity or operation is selected for execution, or
- some scheduling/disposing of an activity, or a subactivity, or an operation is performed.

In the initial configuration (1) the first expression that can be evaluated is `new Bool[5]` whose evaluation produces to the array `[f, f, f, f, f]` (In the array we use `f` for `false` and `t` for `true`.) Then the expression `make Table([f, f, f, f, f])` reduces to an artifact reference α and adds to the configuration the initial artifact instance that follows:

$$\alpha = \langle .\text{isBF} = [f, f, f, f, f], \emptyset, \emptyset, \emptyset \rangle^{\text{Table}}$$

After the initialization of the local variable τ the agent instance γ_{Main} becomes

$$\gamma_{\text{Main}} = \langle \emptyset, \emptyset, \text{main} \left[\begin{array}{l} \boxed{\text{spawn Philosopher}(0, 1, \alpha)}; \\ \dots \\ \text{spawn Philosopher}(4, 0, \alpha) \end{array} \right] \rangle \{ \}^{\text{Main}}$$

The five `spawn` expressions are evaluated from left to right. The evaluation of the expressions `spawn Philosopher(0, 1, α)` reduces to γ_0 and adds to the configuration the agent instance

$$(2) \ \gamma_0 = \langle \emptyset, \sigma_0, \text{main} \left[\begin{array}{l} \boxed{\text{prepare}() \langle \text{true} \rangle}; \\ \text{living}(0, 1, \alpha) \langle \text{memo}(\text{hungry}) \rangle \end{array} \right] \rangle \{ \}^{\text{Phil.}}$$

and the sensor instance $\sigma_0 = \langle \emptyset \rangle^{\text{Sns}}$. Similarly, the reduction of the other `spawn` expressions generates four agent instances and four sensor instances producing the configuration:

$$\gamma_{\text{Main}} = \langle \dots \rangle \ \sigma_0 = \langle \emptyset \rangle \ \dots \ \sigma_4 = \langle \emptyset \rangle \ \alpha = \langle \dots \rangle \ \gamma_0 = \langle \dots \rangle \ \dots \ \gamma_4 = \langle \dots \rangle$$

¹ The syntax of FAL does not include local variables and array object values. In this example, we will handle the local variable τ by replacing, after its declaration/inizialization, all its occurrences with its value.

in which the agent γ_{Main} is inactive, having finished the evaluation of its body. The artifact α does not have any pending operation, and all the agent philosophers may start the execution of the sub-activities of their main activity (by starting the evaluation of the preconditions of `prepare` and `living`). Our modeling make use of nondeterministic evaluation rules, but parallel execution could be modeled.

Going back to (2), since the precondition of the run-time sub-activity `prepare()` of the activity `main` of the agent γ_0 is `true` the expression `prepare() <true>` is replaced by `prepare()[+memo(hungry)]` (whose evaluation causes the insertion of the label `hungry` into the memo of γ_0) and then since the body is fully evaluated `prepare` is removed from the sub-activities of `main`, yielding

$$(3) \gamma_0 = \langle \text{hungry}, \sigma_0, \text{main} \left[\text{living}(0, 1, \alpha) \left[\boxed{\langle \text{memo}(\text{hungry}) \rangle} \right] \right] \{ \} \rangle^{\text{Phil.}}$$

If instead of evaluating the sub-activity `prepare` we would have evaluated the precondition of the sub-activity `living`, the result would have being

$$\gamma_0 = \langle \emptyset, \sigma_0, \text{main} \left[\begin{array}{l} \text{prepare}() \langle \text{true} \rangle, \\ \text{living}(0, 1, \alpha) \langle \text{false} \rangle \end{array} \right] \{ \} \rangle^{\text{Phil.}}$$

Next time the sub-activity `living` was scheduled for execution `living(0, 1, α) <false>` would have been replaced with `living(0, 1, α) <memo(hungry)>`.

Continuing from (3) the precondition `memo(hungry)` of `living` evaluates to `true` and the sub-activity `living(0, 1, α) <true>` is replaced by the corresponding run-time activity resulting in the following:

$$\text{living}(0, 1, \alpha) \left[\begin{array}{l} \text{eating}(0, 1, \alpha) \left[\boxed{\langle \text{memo}(\text{hungry}) \rangle}, \right. \\ \text{thinking}() \langle \text{completed}(\text{eating}) \rangle, \\ \left. \text{shutdown}() \langle \text{failed}(\text{eating}) \rangle \right] \{ \} \end{array} \right]$$

The precondition `memo(hungry)` evaluates to `true` and the sub-activity `eating(0, 1, α) <true>` is replaced by the corresponding run-time activity resulting in the following

$$(4) \text{eating}(0, 1, \alpha) [\left. \begin{array}{l} \boxed{\text{use } \alpha.\text{getForks}(0, 1) : \text{sns } \sigma_0; \\ \text{sense } \sigma_0 : \text{Filter forks_acquired}; \\ /* eat */ \\ \text{use } \alpha.\text{releaseForks}(0, 1); \\ -\text{memo}(\text{hungry}) \end{array} \right\}$$

(Note that both `completed(eating)` and `failed(eating)` would evaluate to `false`.) The evaluation of the body of `eating` can now start by reducing the expression `use α .getForks(0, 1) :sns σ_0` , that schedules the operation `getForks` in the artifact instance α yielding

$$\alpha = \langle .\text{isBF} = [\text{f}, \text{f}, \text{f}, \text{f}, \text{f}], \emptyset, \emptyset, (\sigma_0, \text{getForks} \langle \mathbf{e}'_0 \rangle \{ \mathbf{e}_0 \}) \rangle^{\text{Table}}$$

where \mathbf{e}'_0 is `(not(.isBF[0]) and (not(.isBF[1])))` and \mathbf{e}_0 is

$$.\text{isBF}[0] = \text{true}; .\text{isBF}[1] = \text{true}; \text{signal}(\text{forks_acquired}).$$

The guard \mathbf{e}'_0 reduces to `true`. The reduction of \mathbf{e}_0 updates the array ι to `[t, t, f, f, f]` and adds the label `forks_acquired` to the queue of events of the sensor instance σ_0 , yielding $\sigma_0 = \langle \text{forks_acquired} \rangle^{\text{Sns}}$. Other agents may schedule operation

the artifact α . For instance, if the agent γ_1 and γ_2 invoke the operation `getForks` on α , when the evaluation of `getForks` for the agent γ_0 was completed the state of the artifact would be

$$\alpha = \langle .isBF = [t, t, f, f, f], \emptyset, \emptyset, (\sigma_2, getForks \langle e_2' \rangle \{e_2\}) (\sigma_1, getForks \langle e_1' \rangle \{e_1\}) \rangle^{Table}$$

So the guard $e_1' ((\text{not}(.isBF[1]) \text{ and } (\text{not}(.isBF[2]))))$ would evaluate to `false`, and the associated operation would be rescheduled and put at the rear of the queue yielding the following

$$\alpha = \langle .isBF = [t, t, f, f, f], \emptyset, \emptyset, (\sigma_1, getForks \langle e_1' \rangle \{e_1\}) (\sigma_2, getForks \langle e_2' \rangle \{e_2\}) \rangle^{Table}$$

so the evaluation of the guard of the `getForks` operation invoked by γ_2 may start (and will successfully acquire the forks for γ_2). At the same time, the expression `sense σ_0 :filter forks_acquired` in (4) could be evaluated, perceiving the event `forks_acquired` and removing it from the sensor instance σ_0 which becomes $\sigma_0 = \langle \emptyset \rangle^{Sns}$. The code “`/* eat */`” may be executed and, at the end of its execution the expression `use α .releaseForks(0,1)` schedules the operation `releaseForks` on the artifact α and then `-memo(hungry)` removes the label `hungry` from the memo completing the execution of the sub-activity `eating`. The sub-activity `eating` is discarded and therefore the predicate `completed(eating)` becomes true and the sub-activity `thinking` could be executed resulting in γ_0 to be:

$$\langle \emptyset, \sigma_0, \text{main}[\text{living}(0, 1, \alpha) \left[\begin{array}{l} \text{thinking}() [] \{ \\ /* think */ +\text{memo}(\text{hungry}) \} \\ \text{shutdown}() \langle \text{failed}(\text{eating}) \rangle \end{array} \right] \{ \} \{ \} \} \rangle^{Phil}.$$

(If the evaluation of the predicate `completed(eating)` was done before completion of predicate `eating` the result would have been `false`, and then its evaluation rescheduled.) Once the sub-activity `living` completes its execution, in the example of Fig. 1 it would be rescheduled (since its persistency condition is `true`).

Properties. We have defined a type system for FAL – not reported in the paper for lack of space. The soundness of the type system implies that the execution of well-typed agents and artifacts does not get stuck. The following properties of interaction between well-typed agents and artifacts, which are useful in concurrent programming with SIMPA, hold: (i) there is no `use` action specifying an operation control that is not part of the usage interface of the artifact; (ii) there is no `observe` action specifying an observable property that does not belong to the specified artifact; and (iii) an executing activity may be blocked only in a `sense` action over a sensor that does not contain the label specified in the filter—i.e., the agent explicitly stops only for synchronization purposes. Moreover, a type restriction on sensors – not present in the current type system – may be defined to enforce that there is no `sense` action indefinitely blocked on sensing event e due to the fact that the corresponding triggered operation was not designed to generate e .

4 Related Work and Conclusions

The extension of the OO paradigm toward concurrency — i.e. object-oriented concurrent programming (OOC) — has been (and indeed still is) one of the most important and challenging themes in the OO research. Accordingly, a quite large amount of theoretical results and approaches have been proposed since the beginning of the 80's, surveyed by works such as [4,16,2,11]. We refer to [13] for a comparison of the agent and artifact programming model with *active objects* [9] and *actors* [1] and with more recent approaches extending OO with concurrency abstractions, namely POLYPHONIC C# [3] and JOIN JAVA [8] (both based on Join Calculus [6]). Another recent proposal is STATEJ [5], that proposes *state classes*, a construct for making the state of a concurrent object explicit. The objective of our approach is quite more extensive in a sense, because we introduce an abstraction layer which aims at providing an effective support for tackling not only synchronisation and coordination issues, but also the engineering of passive and active parts of the application, avoiding the direct use of low-level mechanisms such as threads.

In this paper we described FAL, a core calculus to provide a rigorous formal framework for designing agent-oriented languages and studying properties of agent-oriented programs. To authors knowledge, the only attempt that has been done so far applying OO formal modelling techniques like core calculi to study properties of agent-oriented programs and of agent-oriented extensions of object-oriented systems is [13].²Future work concerns a comprehensive study and discussion of the type system, exploring its usage for the analysis of the computational behaviour of agents and artifacts. Properties that we are investigating mainly concerns the correct execution of activities, in particular: (i) there is no activity which are never executed because of their pre-condition; (ii) post-conditions for activity execution can be statically known, expressed as set of memos that must be part of the memo space as soon as the activity has completed; (iii) invariants for activity execution can be statically known, expressed as set of memos that must be part of the memo space while the activity is in execution; (iv) there is no internal action reading or removing memos that has not been previously inserted. We are investigating the suitable definition of pre/post/invariant conditions in terms of sets of memos that must be present or absent in the memo space, so that it would be possible to represent high-level properties related to set of activities, such as the fact that an activity A would be executed always after an activity A' or that an activity A and A' cannot be executed together. On the artifact side, the computational model of artifacts ensures a mutually exclusive access to artifact state by operations executed concurrently; more interesting properties could be stated by considering not only atomic but also structured operations, not dealt in this paper.

References

1. Agha, G.: Actors: a model of concurrent computation in distributed systems. MIT Press, Cambridge (1986)

² The paper revises and extends the formalisation proposed in [13], in particular by formalising a larger set of features, including – for example – the agent *agenda*.

2. Agha, G., Wegner, P., Yonezawa, A. (eds.): Research rections in concurrent object-oriented programming. MIT Press, Cambridge (1993)
3. Benton, N., Cardelli, L., Fournet, C.: Modern concurrency abstractions for C#. *ACM Trans. Program. Lang. Syst.* 26(5), 769–804 (2004)
4. Briot, J.-P., Guerraoui, R., Lohr, K.-P.: Concurrency and distribution in object-oriented programming. *ACM Comput. Surv.* 30(3), 291–329 (1998)
5. Damiani, F., Giachino, E., Giannini, P., Drossopoulou, S.: A type safe state abstraction for coordination in java-like languages. *Acta Inf.* 45(7-8), 479–536 (2008)
6. Fournet, C., Gonthier, G.: The reflexive chemical abstract machine and the join calculus. In: *POPL 1996*, pp. 372–385. ACM, New York (1996)
7. Igarashi, A., Pierce, B., Wadler, P.: Featherweight Java: A minimal core calculus for Java and GJ. *ACM TOPLAS* 23(3), 396–450 (2001)
8. Itzstein, G.S., Kearney, D.: Join Java: an alternative concurrency semantics for Java. Technical Report ACRC-01-001, Univ. of South Australia (2001)
9. Greg Lavender, R., Schmidt, D.C.: Active object: an object behavioral pattern for concurrent programming. In: *Pattern languages of program design*, vol. 2, pp. 483–499. Addison-Wesley Longman Publishing Co., Inc., Boston (1996)
10. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 19 (2009); Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems
11. Philippsen, M.: A Survey of Concurrent Object-Oriented Languages. *Concurrency Computat.: Pract. Exper.* 12(10), 917–980 (2000)
12. Ricci, A., Viroli, M.: SIMPA: An agent-oriented approach for prototyping concurrent applications on top of java. In: *PPPJ 2007*, pp. 185–194. ACM, New York (2007)
13. Ricci, A., Viroli, M., Cimadamore, M.: Prototyping concurrent systems with agents and artifacts: Framework and core calculus. *Electron. Notes Theor. Comput. Sci.* 194(4), 111–132 (2008)
14. Ricci, A., Viroli, M., Piancastelli, G.: SIMPA: An agent-oriented approach for programming concurrent applications on top of java. *Science of Computer Programming* (2010), doi:10.1016/j.scico.2010.06.012
15. Sutter, H., Larus, J.: Software and the concurrency revolution. *ACM Queue: Tomorrow's Computing Today* 3(7), 54–62 (2005)
16. Yonezawa, A., Tokoro, M. (eds.): *Object-oriented concurrent programming*. MIT Press, Cambridge (1986)

Using Trace to Situate Errors in Model Transformations

Vincent Aranega¹, Jean-Marie Mottu², Anne Etien¹, and Jean-Luc Dekeyser¹

¹ LIFL - UMR CNRS 8022, INRIA - University of Lille 1
Lille, France

² LINA - UMR CNRS 6241 - University of Nantes
Nantes, France

{vincent.aranega, anne.etien, jean-luc.dekeyser}@lifl.fr,
jean-marie.mottu@univ-nantes.fr

Abstract. Model Driven Engineering (MDE) promotes models as main artifacts in software development process. Each model represents a viewpoint of a system. MDE aims to automatically generate code from an abstract model, using various intermediary models. Such a generation relies on successive model transformations shifting a source model to a target one. The resulting transformation sequence corresponds to the skeleton of an MDE based approach, similarly to compiler in traditional ones.

Transformations are used many times in order to justify their development effort. If their are faulty, they can largely spread errors to models. Thus, it is indispensable to test them and possibly debug them. In this paper, we propose an error localization algorithm based on a traceability mechanism in order to ease the transformations debugging. We illustrate this approach in the context of embedded system development.

Keywords: Model transformation, Error localization, Metamodels, Traceability, Tests.

1 Introduction

Model Driven Engineering (MDE) promotes models as main artifacts in the life cycle of complex systems. A key component of MDE is the definition and application of model transformation that shift a source model to a target one. These transformations are composed of rules and are defined in terms of metamodels relative to the source and the target languages. Transformations can be chained if the source metamodel of one of them is the target metamodel of another one. Thus model transformations form the skeleton of the system development.

Using traditional approaches, errors observed in the execution of the system may come from the compiler or the source program. In an MDE approach, such a distinction can be established, between errors in the transformation definition and errors in the source model. Errors in transformation may have huge consequences. Indeed, transformations are used many times to justify the efforts relative to their development. So if they are erroneous, they can spread fault to models several times. Furthermore, systems may evolve, leading to errors and implying changes in different sub-parts to lead to a new stable configuration. Even if these issues are common to any system, they require a specific management when a model driven development approach is used.

Traceability is potentially relevant to help designers to solve these issues. The trace is usually used to link the requirements to the implementation artifacts. However, traceability also establishes relationships between products of a development process, especially products bound by a predecessor-successor or master-subordinate relationship [1]. Regarding MDE and more specifically model transformations, the traceability mechanism links elements of different models in order to specify elements useful to generate others. Those links can also be used to analyze impacts of model evolutions onto other models in the transformations chain. Finally, it is reasonable to consider traceability as a bridge between the business and the transformation world, if the transformation parts are explicitly associated to the links. Business is so materialized by the models useful to generate the system.

We have already defined a traceability algorithm based on two metamodels [2]. One captures the trace relative to a single transformation, whereas the other manages the relationships all along the transformation chain. These metamodels are rich enough to support algorithms dedicated to the resolution of the previously cited issues. In this paper, we focus on error localization in model transformation and we propose an algorithm based on our traceability metamodels. From a given generated element, our approach identifies the rule sequence of each intermediate transformation of the complete transformation chain. First, tests identify incorrect parts of the produced model, then, the erroneous rule which causes this failure is detected. We have successfully applied this algorithm on a case study in the context of embedded system development.

This paper is organized as follows. Section 2 presents existing traceability solutions in MDE. Section 3 gives different ways to exploit trace. In section 4, the trace metamodels are introduced and the use of the trace models for faulty transformation rules identification is described. Section 5 illustrates our approach with a case study based on a QVT transformation. Finally, we conclude the paper and suggest future works in section 6.

2 Related Work

In MDE, many solutions for traceability are proposed in the literature [3], [4], each of them responding to specific needs of projects.

MDE has as main principle that *everything is a model*, so the trace information is stored as models [5]. Classically, two main approaches exist. The first focuses on the addition of trace information on the source or target model [6]. The major drawback of this solution is that it pollutes the models with additional information and it requires the metamodels adaptation in order to take into account traceability. However it produces the most comprehensive traces. The second solution focuses on the storage of the trace as an independent model. Using a separate trace model with a specific semantics has the advantage of keeping trace information independent of original models [5]. To deal with the advantage of these two techniques, a solution consist on the possibility to merge on-demand the trace model with the transformation source or target model [7].

Several metamodels have been proposed as the foundation of traceability approaches [5], [8], [9]. These metamodels are structurally, relatively different and often depend on a specific transformation language. However, they always gather the same basic information. The traceability metamodel defined by Jouault [5] only contains minimal

information, *i.e.* elements and links between them. The name of the transformation rule which creates the elements is associated to the link as a property. Based on these works, Yie *et al.* propose an advanced trace for ATL [10]. The proposed trace gathers fine grained information of the ATL transformation execution. However, the implementation is really dependent from the ATL metamodel and the ATL Virtual Machine. Our local trace metamodel is an extension of the one defined by Jouault “core” trace providing a more finer grain trace and separating the rule concept from the link to ease manipulations.

Collecting the trace information can be easily performed during the transformation execution since this only incurs a small cost [9]. Indeed the trace model is thus viewed as an additional target model. For this reason, trace generation could be manually implemented in transformations to produce an additional trace target model or it can be supported by the transformation engine [11]. In [5], an automatic generation of trace code into rule code is presented, based on the fact that transformation programs are models that could be transformed into another model that contains trace code. Nevertheless, these solutions impose to inject code in transformation rules or transformation engine. To remain the less intrusive as possible, Amar *et al.* propose another technique using aspect programming [12]. Regrettably, for the moment, this solution cannot be used with every transformation languages.

Once the trace is generated, the main interest for the user is to have access to the information he needs. However, in case of transformations chain, the trace models relying only on the two concepts *Element* and *Link*, which are produced during the transformations, are not enough. One solution is to add the concept of *Step*, referring to a transformation, in the trace model such as in the trace mechanism of Kermeta [8]. Traceability links are gathered by step (*i.e.* by transformation) what thus allows to manage transformation chains. An other solution is to externalize the navigation between initial models and trace models of a whole transformation chain in another model, called megamodel. It refers to the traceability in the large, whereas model to model transformations refer to a traceability in the small [13].

3 Using Traceability in Model Driven Engineering

In the introduction, we identified three different issues that can be encountered in the development of complex systems: fix the system itself, fix the transformations generating the system and manage the impact of evolutions on the whole system. We have suggested that a traceability mechanism can solve these issues. In this section, we show that, while remaining in MDE, each of these purposes requires different traceability information.

3.1 Needed Trace Information

When an error in the generated system is found or when an unexpected behavior is observed, the system has to be fixed. For this purpose, the elements of the input models that engender the (or one of the) incorrect element(s) have to be identified. Such information are at the heart of any traceability mechanism and are materialized by links between source elements and target elements. These can also be used to analyze the impact of the

input model evolutions on the output model and to propagate these changes. However, when the system evolves, the transformation may also evolve. To overcome this issue, the rule engendering a traceability link must be associated to it. Furthermore, the traceability mechanism has to be adapted to the model driven development reality. Indeed, complex systems do not rely on a single transformation but on one or several transformation chain. Therefore traceability should support relationships all along the chain.

As a first conclusion, we have demonstrated that links between source and target elements are not enough to build an efficient traceability mechanism dedicated to system fixing and system evolution. Information relative to the transformation rules and allowing the navigation in the transformation chain are required.

It can be noticed that fixing the system can be performed only if we are confident in the transformations that generate it. In the following subsection, we focus on using traceability in transformation test and show that information relative to the input/output elements relationships, transformation rules and navigation in the transformations chain are, in that case, also required.

3.2 Trace Exploitation for Model Transformation Testing

In this subsection, we briefly present the transformation test and then we show how trace can be used in this context.

Model Transformation Testing. By automating critical operations in system development, model transformations are time and effort saving. However, they may also introduce additional errors if they are faulty. Therefore, systematic and effective testing of transformations is necessary to prevent the production of erroneous models.

Several problems need to be solved when tackling model transformation testing. First, we need to detect the presence of errors by observing wrong execution of the model transformation. Corresponding challenges are efficient test data production and observation of error in the system. We then have to locate the error in the transformation and to fix it. Figure 1 sketches the test transformation process and associates its different parts to the corresponding test problematics.

Efficient test data production and error observation are challenges out of this paper scope. Nevertheless, we briefly illustrate them. In this paper, we focus on error localization.

Transformations manipulate models, which are very complex data structures. This makes the problems of test data generation, selection, and qualification, as well as error observation very difficult.

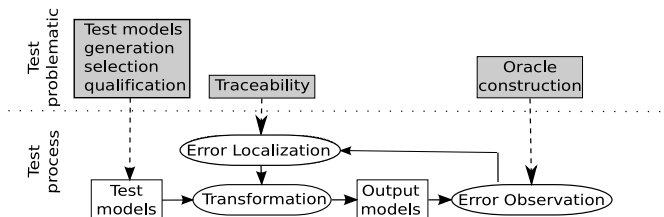


Fig. 1. Test transformation process

Test data generation consists in building models conform to the input metamodel. Their number is potentially infinite so the first challenge is to define criteria for test data generation [14]. Then, the resulting test models set has to be qualified, depending on their coverage of the input domain or on their ability to detect potential errors.

Error observation relies on the detection of an error in a model produced by the transformation. In [15] we proposed an approach based on the construction of oracles. The oracle checks the validity of the output model resulting from the transformation of one test model. It relies either on properties between elements of the input and output models or properties only concerning the output model. These properties have to be formalized and must cover the whole metamodels. Defining oracles is difficult since human intervention is required. Indeed, extracting information to produce oracle from the model transformation requirements cannot be automatized.

Error Localization in Model Transformation. Errors observed in the output model can concern: wrong property value, additional/missing class, etc. They result from errors in the transformation. Where are they and what are they, are two questions that remain unanswered.

The error can be everywhere in the transformation. Its detection is easier if the search field is reduced to the faulty rule, *i.e.* the rule that creates the incorrect element (or doesn't create an expected element) in the output model. Once the error localized in the transformation, in order to fix it, the input model elements leading to this incorrect output element have to be identified.

Finally, due to the non exhaustiveness of test and the complexity of building oracles, test of a single transformation can be missed at the expense of test of the whole transformation chain.

4 Traceability Metamodels Description

To solve the problems we want manage (*e.g.* system debugging, transformation debugging, design alternative exploration...), we have defined our own trace approach [2]. This approach provides a traceability in the small and in the large [13], which we refer as local and global traceability respectively. It relies on two metamodels: the Local Trace metamodel corresponding to the model to model traceability and the Global Trace metamodel helping in the global navigation. These two metamodels are completely independent from the transformation language and can even be used with various languages. Only the trace generation changes. Our traceability mechanism allows users to trace elements all along a transformation chain where each transformation may be written in different languages.

4.1 Local Trace Metamodel

The Local Trace metamodel is used to capture the traces between the input and the output of one transformation. The metamodel is based on the trace metamodel presented in [5]. Figure 2 shows the Local Trace metamodel.

The Local Trace metamodel contains two main concepts: *Link* and *ElementRef* expressing that one or more source elements are possibly bound to target elements. Those

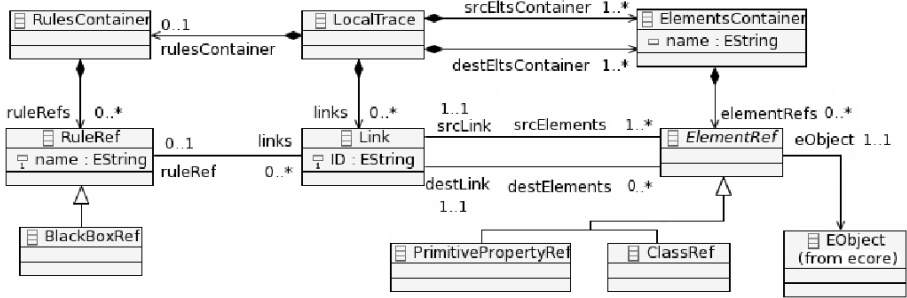


Fig. 2. Local Trace Metamodel

concepts are the same as in [5]. All the other concepts have been added to provide a finer and more complete trace. In our metamodel *ElementRef* is an abstract class representing model elements that can be traced (*i.e.* properties and classes). It owns a *name* attribute catching the classes and properties names (if they exist in the traced models). Property values referring to a primitive types like Integer, Double, String etc. are traced using the *PrimitivePropertyRef* concept. It gathers the property value (using the *value* attribute) and the property type (using the *type* attribute). Properties typed by a class are traced by *ClassRef*.

More information is needed in order to trace the transformation rules and black-boxes. The rule producing the link is traced using the *RuleRef* concept. A rule can be associated to several links, so the association is many to one between *RuleRef* and *Link*. The *RuleRef* concept is optional and doesn't need to be generated if it is not used. In case of error localization such information is definitively useful. Black-Boxes are special kind of rules: producing some output model elements from input model elements. So, they can be traced with *Link*. The treatment performed by a black-box may be externalized (such as a native library call) but in every case is opaque to the designers. We take care to differentiate black-boxes and rules since test only deals with rules. The *BlackBox* concept is a subclass of *RuleRef*. Both establish a bridge with the transformation world.

An *ElementRef* refers to the real element (*EObject*) of the input (resp. output) model instantiating the *ECore* metamodel. The *LocalTrace* concept represents the root of the Local Trace model. It contains possibly one *RulesContainer* and several *ElementsContainers* (one for each source (respectively destination) models), gathering *RuleRefs* and *ElementRefs*, respectively. Separating sources and targets elements helps in reducing the cost of search of input or output elements.

4.2 Global Trace Metamodel

The Global Trace model [2,13] links together the local traces following the transformation chain. Thus, the Global Trace model ensure the navigation from local trace models to transformed models and reciprocally as well as between transformed models. The global trace can also be used to identify the local trace associated to a source or destination model.

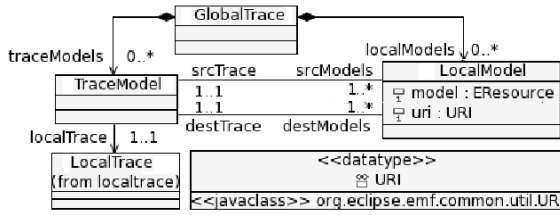


Fig. 3. Global Trace Metamodel

It also provides a clear separation of trace information, which leads to a better flexibility for trace creation and exploitation. Without this global trace all traceability links of the whole transformation chain are gathered in a unique trace model.

Figure 3 shows the global trace metamodel. Each *TraceModel* produced during a transformation and referring to a *LocalTrace*, binds two sets of *LocalModels*. These are shared out transformations, indicating that they are produced by one transformation and consumed by another. The *GlobalTrace* concept here represents the root of the model.

4.3 Trace Generation

The proposed metamodels are completely language independent. However trace generation requires information contained in the transformation and so relies on the transformation language. Whatever the transformation language, the trace generation is a two steps algorithm. The first step corresponds to the production of a local trace for each transformation and the second, to the generation of the global trace specifying the transformation chain.

In the following, we only focus on the trace generation from transformation written in QVTO [16], an implementation of the standard QVT language [17].

The local trace generation has to be, if possible, non-intrusive in the transformation code or in the engine. The execution of the QVTO transformations uses a trace mechanism to store a mapping between model elements and to resolve reference. This trace is relatively complex and dedicated to the transformation execution. However, it gathers the information useful to generate the local trace models conformed to our local trace metamodel. In particular, it refers the source elements, their associated target elements and the rule used to produce the latter from the former. The produced QVTO trace is transformed into a local trace.

The global trace production is based on information relative to the generated local traces. From the local traces, the transformation sequence can be rebuilt. Indeed, the models never appearing as output models in any local traces are the start models. From these models and traces, the other can be deduced.

If the transformation languages evolve, only the local trace generation may be impacted. Indeed, this latter directly relies on the used transformation language, whereas the global trace is build from the local traces.

4.4 Error Localization Algorithm

Our error localization algorithm requires that an error has been beforehand observed in an output model. The transformation producing this model contains errors. Our algorithm aims to reduce the investigation field by highlighting the rule sequences which lead to the observed error.

Our algorithm is based on the following hypothesis. Let us consider two elements A and B of the output model created by the rules $toA()$ and $toB()$ respectively. If A references B through an association, it assumes that the rule $toA()$ calls the rule $toB()$ or makes an operation to reference B .

In case of an erroneous property (e.g. with an unexpected value) in an element, the faulty rule is easily identified. It corresponds to the *RuleRef* coupled to the *Link* associated to the *ElementRef* referring the selected element. In case of an error on an element (e.g. added or missing), the faulty rule is one which calls the last rule involved in the creation of the selected element. Causes can be a missing or misplaced rule call.

We detail the algorithm in the second case:

1. select the faulty element and identify the model to which it belongs
2. from the Global Trace model, recover the Local Trace model whose the previously identified model is one of the output models
3. look for the *ElementRef* corresponding to the selected element in the local trace *destContainer*
4. recover the *RuleRef* associated to the *ElementRef* by navigating through the trace links,
5. store the *RuleRef* and the *eObject* type
6. search, in the *destContainer*, the *ElementRef* which have their *eObject* linked by an association to the *eObject* corresponding to the *ElementRef* identified in step 3
7. apply recursively the algorithm from step 3 on each element found in step 4

The recursive call stops when no direct linked *eObject* can be found in step 6. The rule is called by no other one; it is an entry point of the transformation. Technically, it is materialized by the storage of a *null* pointer.

Thus, the algorithm results in a kind of tree representing the successions of rules producing the selected element. It has been applied with success on transformations written with different transformation languages (QVTO and a Java API).

5 Case Study

Debugging transformations, even if they are simple, is often a tough job. As soon as we operate a scale-up, this task becomes unmanageable. In this section, we illustrate how our approach eases transformation debugging in the Gaspard2 environment by automating the error localization.

5.1 Overview

Gaspard [18] is a co-design environment for Embedded Systems. In this environment, the hardware architecture and the application are separately designed at a high level of

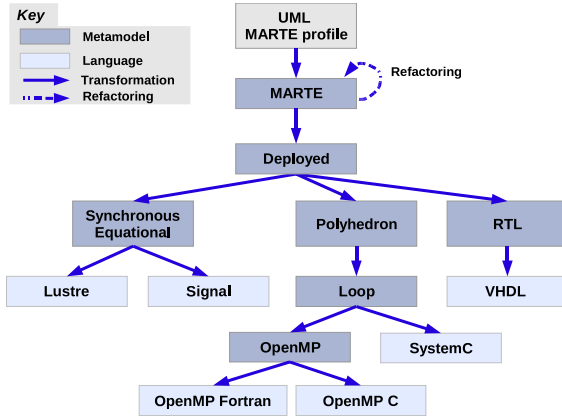


Fig. 4. MDE skeleton of Gaspard

abstraction using UML enriched with the MARTE profile [19] dedicated to modeling and analysis of real time and embedded system. In order to generate code that will be used for hardware-software co-simulation, functional verification or circuitry synthesis, several intermediate metamodelling representing different levels of abstraction have been specified. Each metamodel introduces new concepts more platform-dependent. Transformations between these metamodelling have been written in order to automatically produce intermediate models and generate code. Thus several transformations chains have been defined; one per targeted platform. Figure 4 shows an overview of the MDE skeleton of the Gaspard environment by specifying the metamodelling and languages in presence and the transformations between them [20].

In this case study, we only focus on a single transformation from the MARTE metamodel to the Deployed metamodel. This transformation is written with QVTO. The MARTE metamodel contains around 80 metaclasses whereas the output metamodel is in fact decomposed into five metamodelling and contains around 60 metaclasses.

The main idea is to test this transformation on an input test model and, for example using an oracle in order to observe an error on the produced model. The oracle checks, among others, that any model produced by the transformation from the MARTE to the Deployed metamodel has a unique root. This root is a *DeploymentSpecificationModel* instance has been produced from an instance of the MARTE *Model* metaclass.

5.2 Illustration of the Localization Algorithm

Using our localization algorithm and from the error reported by the oracle, we can debug more precisely the transformation. First, the models corresponding to the local and the global traces, have to be generated.

The top of Figure 5 shows a fragment of the output model. It contains several roots whose some (the *PortImplementedBy*) are not instance of the *Downscaler:DeploymentSpecificationModel*. An error on an element is thus detected in the transformation. We apply our error localization algorithm on one of the output model “misplaced” elements in order to highlight the rule sequence and identify the faulty rule.

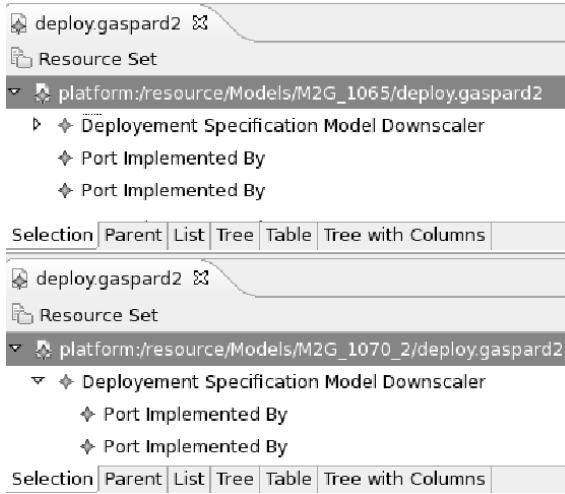


Fig. 5. Excerpt of The generated output model(top) and of the manually produced model (bottom)

Figure 6 shows a sketch of the output model and its associated traces. The algorithm begins with the selection of a *PortImplementedBy* element. For example, we select the *pi1:PortImplementedBy* element that belongs to the *deploy.gaspard2* model. The local trace associated to this model is recovered using the global trace model.

In the *lt1:LocalTrace*, *cr2:ClassRef:ElementRef* corresponds to the *pi1:PortImplementedBy*. Navigating through the *l2:Link* associated to *cr2:ClassRef*, the *toImplementedBy:RuleRef* rule is identified and stored. Then, *ElementRefs* are scanned to identify elements linked to *pi1:PortImplementedBy*. Here, neither the *deploy.gaspard2* model nor other models produced by the transformation own elements linked through an association (including compositions) to the *pi1:PortImplementedBy*. So, this step returns nothing, the *toImplementedBy:RuleRef* rule is not called by another rule. Thus a *null* pointer is stored and the algorithm stops executing. The produced rule call tree contains only two elements: the *RuleRef* named *toPortImplementedBy* associated to the type of the eObject on which it is applied; (*PortImplements*) and the *null* pointer.

In Figure 7 we present a piece of the QVTO transformation code illustrating that the rule is called by the main function (line 77). The code of the rule itself corresponds to line 1698 to 1705.

The precedent analysis leads to the conclusion that the *toPortImplementedBy* rule may be called by another rule or a reference is missing in a rule. Further analysis can be done by manually specifying an expected output model (top of Figure 5) corresponding to the input model. Comparing the generated output model to this new one, we can see that the *DeploymentModel* contains *PortImplementedBy* elements. So the rule call should be moved from the *main* entry point to the rule which creates the *DeploymentModel* element.

The example developed here is quite simple, but illustrates the easiness to identify a faulty rule in a huge transformation (more than 2000 lines of code). This algorithm has to be used in the context of a transformation test. It requires the results of the test

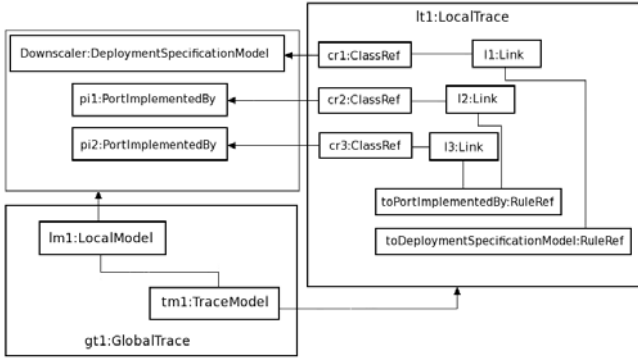


Fig. 6. Excerpt of the output model, of the local and the global trace models

```

52. main() {
...
77.  deploy.objects()[ImmDeployment::PortImplements].map
      toPortImplementedBy();
...
80. }
...

1698. mapping ImmDeployment::PortImplements::toPortImplementedBy():
      Gaspard2::PortImplementedBy @gaspardDeploy
1699.   when {
1700.     self.target.resolve(Gaspard2::Port)->notEmpty()
1701.   }
1702.   {
1703.     port := self.target.resolveone(Gaspard2::Port);
1704.     portImplementation := self.source
      resolveone(PortImplementation);
1705.   }
...

```

Fig. 7. QVTO Transformation Excerpt

generation and the errors observation steps. It reduces the field of potential faulty rule to the only rules involved in an element creation. Thus, using this approach, we have reduced the search field for the previous example to one rule.

5.3 Error Localization in Transformation Chain

The algorithm presented in section 4.4 is dedicated to error localization in a single transformation, but we develop a variation adapted to transformation chain. Not only the successive rules are stored but also any element of the input model that was useful to the creation of the faulty output element. The algorithm is then again applied on each of these elements. The final result is a set of rules corresponding to the set of potential faulty rules on the whole transformation. For sake of space we do not illustrate this algorithm which has nevertheless been successfully implemented and used with the Gaspard transformation chain.

6 Conclusions and Future Works

In this paper, we have proposed a traceability based mechanism to locate errors in a single model transformation or a transformation chain. It reduces the investigation field to the rules called to create an output element identified as erroneous in a preliminary test phase. The localization is based on three main parts: an error observed in an output model, our trace models and the localization algorithm. The error can be point out by an oracle whereas the traces give the support for the localization algorithm.

As the algorithm is based on our traces metamodels, it is purely language independent and can be reused for any transformation languages as long as the local and the global trace are generated. For the experimentation, we use our approach on transformation written in QVTO. It has also been successfully tested on transformations using a dedicated Java API. Our approach has shown its efficiency on the transformation chains of the Gaspard framework. We have also shown in [21], that the trace provided by default with QVTO and defined in the QVT specifications is not adapted in lot of contexts, for example, when information concerning properties are needed.

Currently, the localization gives a set of potential faulty rules. To exactly determine the faulty rule, the set returned by the algorithm must be manually analyzed. This final step can be automatized by introducing new oracle answers. Indeed, with some additional information, we could, little by little, reduce the search field to a faulty rule and find the rule to modify.

Based on these results we have started works on mutation analysis [22] in order to test model transformation [23]. Mutation analysis aims to validate a set of test input model. It relies on the identification of fault voluntary injected in transformations. Indeed, variants of a transformation to test are created by injecting a single error each time. The output models generated by the original transformation and its variants are compared. If no difference is observed, a new input model has to be created. Traceability mechanism and the algorithms developed in this paper help to create this new model by anticipating what should be the difference observed in the output models.

Acknowledgements. This work was supported by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the 'Contrat de Projets Etat Region (CPER) 2007-2013'.

References

1. IEEE: IEEE standard computer dictionary : a compilation of IEEE standard computer glossaries. IEEE Computer Society Press, New York (1991)
2. Glitia, F., Etien, A., Dumoulin, C.: Traceability for an MDE Approach of Embedded System Conception. In: ECMDA Tracibility Workshop, Germany (2008)
3. Galvao, I., Goknil, A.: Survey of traceability approaches in model driven engineering. In: The Eleventh International IEEE EDOC Conference (EDOC 2007), pp. 313–324. IEEE Computer Society Press, Los Alamitos (2007)
4. Reshef, A.N., Nolan, B.T., Rubin, J., Gafni, S.Y.: Model traceability. *Ibm Systems Journal* 45 (2006)
5. Jouault, F.: Loosely coupled traceability for atl. In: ECMDA Workshop on Traceability (2005)

6. Velegrakis, Y., Miller, R.J., Mylopoulos, J.: Representing and querying data transformations. In: Proceedings of the International Conference on Data Engineering, ICDE, pp. 81–92. IEEE Computer Society, Washington (2005)
7. Kolovos, D.S., Paige, R.F., Polack, F.A.: On-demand merging of traceability links with models. In: ECMDA Workshop on Traceability, Bilbao, Spain (2006)
8. Falleri, J.R., Huchard, M., Nebut, C.: Towards a traceability framework for model transformations in kermeta, HAL - CCSD - CNRS (2006)
9. Vanhooft, B., Ayed, D., Van Baelen, S., Joosen, W., Berbers, Y.: Uniti: A unified transformation infrastructure. In: MoDELS, pp. 31–45 (2007)
10. Yie, A., Wagelaar, D.: Advanced traceability for ATL. In: 1st International Workshop on Model Transformation with ATL (MtATL 2009), Nantes, France (2009)
11. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal 45(3), 621–646 (2006)
12. Amar, B., Leblanc, H., Coulette, B.: A Traceability Engine Dedicated to Model Transformation for Software Engineering. In: ECMDA Traceability Workshop, Berlin, pp. 7–16 (2008)
13. Barbero, M., Didonet, M., Fabro, D., Bézivin, J.: Traceability and provenance issues in global model management. In: ECMDA Traceability Workshop (2007)
14. Fleurey, F., Baudry, B., Muller, P.-A., Le Traon, Y.: Towards dependable model transformations: Qualifying input test data. SoSyM (2007)
15. Mottu, J.M., Baudry, B., Le Traon, Y.: Model transformation testing: oracle issue. In: MoDeVva Workshop Colocated with ICST 2008, Norway (2008)
16. Borland: Qvt - o (2007), <http://www.eclipse.org/m2m/qvto/doc>
17. Object Management Group, Inc.: MOF Query / Views / Transformations (2007), <http://www.omg.org/docs/ptc/07-07-07.pdf>; OMG paper
18. DaRT Team: Graphical Array Specification for Parallel and Distributed Computing (GASPARD2) (2009), <http://www.gaspard2.org/>
19. Object Management Group: A UML profile for MARTE (2007), <http://www.omgmarTE.org>
20. Gamatié, A., Le Beux, S., Piel, E., Etien, A., Ben Atitallah, R., Marquet, P., Dekeyser, J.-L.: A Model Driven Design Framework for High Performance Embedded Systems. Technical Report 6614, INRIA (August 2008)
21. Aranega, V., Mottu, J.-M., Etien, A., Dekeyser, J.-L.: Using an alternative trace for qvt. In: Multi-Paradigm Modeling, Oslo, Norway (2010)
22. DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on test data selection: Help for the practicing programmer. Computer 11(4), 34–41 (1978)
23. Aranega, V., Mottu, J.-M., Etien, A., Dekeyser, J.-L.: Using traceability to enhance mutation analysis dedicated to model transformation. In: MoDeVva, Oslo, Norway (2010)

Design of SOA Services: Experiences from Industry

Susanne Patig

University of Bern, Institute of Business Information Systems
Engehaldenstrasse 8, CH-3012 Bern, Switzerland
susanne.patig@iwi.unibe.ch

Abstract. Because of the unknown usage scenarios, designing the elementary services of a service-oriented architecture (SOA), which form the basis for later composition, is rather difficult. Various design guidelines have been proposed by academia, tool vendors and consulting companies, but they differ in the rigor of validation and are often biased toward some technology. For that reason a multiple-case study was conducted in five large organizations that successfully introduced SOA in their daily business. The observed approaches are contrasted with the findings from a literature review to derive some recommendations for SOA service design.

Keywords: Service-oriented architecture (SOA), Service design, Case study.

1 Motivation

Service-oriented architecture (SOA) has quantifiable benefits: For example, SWISSTOPO, the Swiss federal geo-information centre, provides map information as Web services (e.g., <http://www.ecogis.admin.ch/>). According to cost estimates, developing new GIS (Geographic Information System) applications using these Web services requires 20 person days of development effort and causes 15,000-30,000 Swiss francs operating costs per year – compared to 200-400 person days and 200,000 – 500,000 Swiss francs for the development of the same GIS application without the Web services.

To realize these benefits, the services of an SOA must be properly designed. But, finding the appropriate SOA design approach is difficult. Therefore it is the aim of this paper to classify, compare and consolidate the distinct ways to design the (elementary) services of an SOA and to give some recommendations for the initial service design. The later composition of these elementary services (e.g., by BPEL engines) is not the topic of this paper.

As any design approach is influenced by the underlying understanding of SOA, we derive the SOA definition used here in Section 2. Based on a literature review, Section 3 sketches and compares approaches to design SOA that have been proposed by practice and academia. Academic approaches do not necessarily work outside the scientific ‘clean room’, and practical approaches usually aim at selling tools or consulting. For that reason we have conducted an independent multiple-case study of five large organizations that successfully introduced SOA in their daily business (see Section 4). The number and deliberate distinction of the cases guarantee the validity of our conclusions in Section 5.

2 Defining Service-Oriented Architecture

Based on a literature review, this section aims at deriving a representative definition of SOA. Only validated (see Section 3) academic SOA design approaches were considered; moreover, we focused on materials in English¹. The academic literature mostly piggybacks on SOA statements proposed in *practice*. Here, major *analysts* [26], pioneering *practitioners* [5], large SOA *software vendors* [1], [8] and [25], *consulting companies* [2] as well as *standards* [14], [17] represent reliable sources.

SOA is frequently defined by referring to its main constituents, the *services*, and several attributes or supplements known as *SOA design principles*. A *design principle* is a generalized, accepted (industry) best practice to build some kind of software solution [4]. For SOA, the following design principles (see also [4]) are applied:

1. *Abstraction* ([26], [5], [16]): Services hide information on technology and logic from the outside world.
2. *Standardized contract* (e.g., [26], [5], [14], [1], [8], [2], [12]): Services provide a technical interface by which they can be accessed, which serves as a contract between the service and its client(s) and keeps to some standard definition language. At the very least, an *interface* consists of operations and their signatures.
3. *Loose coupling* ([5], [3], [25], [21]): A defined service contract is independent of both the service's implementation and the service consumer.
4. *Cohesion* [21]: The functionality provided by a service is strongly related.
5. *Reusability* ([26], [5], [17], [1], [12], [13], [3], [21]): The functionality encapsulated by a service is sufficiently generic for unanticipated usage scenarios.
6. *Composability* ([5], [16], [8], [12]): New services can be built ('composed') from existing ones.
7. *Statelessness* ([5], [2]): Between consecutive service calls, no information must be kept within the service.
8. *Autonomy* ([5], [12], [13], [21]): A service implementation has a significant degree of control over its environment, and services are independent of each other.
9. *Discoverability* ([26], [5], [14], [1]): Services are supplemented by meta data by which they can be found and interpreted.
10. *(Business) Alignment* (e.g., [16], [1], [13], [3]): The functionality provided by a service contributes to business.

The listing above represents the superset of SOA design principles. Closer examination reveals that some principles are related: *Contracts* (interfaces), whether standardized or not, imply *abstraction*. *Abstraction* is in the tradition of 'information hiding': Not only the interface as a whole, but each operation signature masks implementation [19]. Moreover, *abstraction* is a prerequisite to *reusability*, and *composition* is a special form of reuse. *Autonomy* in the sense of maximum independence between services is achieved if the functionality of each service is *cohesive*, and autonomy

¹ An exception is made for [2], as this is the only available SOA design approach of a vendor-neutral consulting company.

facilitates reuse. Finally, *loose coupling* expects *discoverability*. Consequently, the ‘minimal’ set of disjoint SOA principles comprises standardized contract, loose coupling, statelessness, autonomy, and (business) alignment.

Abstraction, contract, autonomy, reusability and statelessness immediately follow from the definition of a *software component* as a unit of composition with contractually specified interfaces and explicit context dependencies only. Components are given in binary form, can be deployed, produced and acquired independently and are subject to composition by third parties [27]. As opposed to objects, components do not have a persistent state (*statelessness*) [27].

Historically, component models (such as CORBA or Enterprise JavaBeans) defined standards for the definition, interaction and composition of components. But, the standards of distinct component models were incompatible [9]. A goal of *Web service (WS)* technology was *standardization* - of interfaces with the WSDL [30] and REST (Representational State Transfer [7]) and of contracts with the complete WS-* stack of specifications [28]. This standardization sets SOA services apart from software components.

Consequently, we use the following definition for our work: *Service-oriented architecture (SOA)* consists of a set of software components that provide their functionality via interfaces as services. Services are design objects in their own right, described with some standard definition language and independent of both the service’s implementation and the service consumers (*loose coupling*). The services should contribute to business.

3 State-of-the-Art of SOA Service Design

Our focus is the design of elementary services for a service-oriented architecture. Thus, papers dealing with the composition of more complex services out of existing ones or service modeling as well as design approaches for specific service properties (e.g., security) are disregarded.

Instead of aiming for completeness, our review tries to gather the commonalities and specifics in designing services for SOA over the distinct sources of such approaches (academia, practitioner, vendor, consulting). By *approach* we mean a systematic way to achieve some goal.

Only validated SOA service design approaches were considered. In order of decreasing strength of evidence, the following types of validation exist (see Table 1): validation by widespread SOA tools or SOA-based software, validation by application in (recurring) consulting practice, validation by application in a (singular) industrial case and validation by a (somehow constructed) sample. Table 1 summarizes the results of the literature review.

Practical approaches to design services for SOA are either based on patterns or hierarchical, whereas academic approaches are always hierarchical. *Best practices* [25] or *SOA patterns* (e.g., [8]; shaded in Table 1), a special form of best practices, are proven solutions that guarantee that some designed software adheres to the SOA design principles (see Section 2). They are helpful in making detailed architectural decisions, evaluating or implementing services. However, only three patterns (‘utility abstraction’, ‘entity abstraction’, ‘task service model’ [5]) deal with ‘discovering’

Table 1. Reviewed SOA service design approaches

	Vendors			Consulting	Academia				
	IBM [1]	Oracle [8, [29]	SAP [25]	sd&m [2]	[12]	[13]	[3]	[20]	[33]
Scenario	All	Mainly W	All	All	All	All	All	All	R
Direction	Mainly TD	Mainly BU	Mainly TD	TD	Mainly TD	Mainly TD	Hybrid	TD	BU
Process	Yes	—	—	Yes	Yes	Yes	Yes	Yes	Yes
Validation	SOA Tools (and consulting)		Software	Consulting	Industry case		Example		
I. Service Identification									
Goals	(1) Scoping	—		(2) Single goals	—	—	—	(1)	—
Functional Areas	(2) Hierarchical decomposition	—		(1) Hierarchical decomposition	(1) Scoping	(Scoping)	—	—	(2) Needed functions
Processes	(3) Activities, control flow	BPEL, Controller	X	(Optional)	(3) Activities, control flow	(1) Activities	(1) Activities	(2)	—
Other	(5) Business rules, variations	Façade, Business rules, Events	Industry Standards	(2) Single roles	(2) Stakeholder	(3) Roles	—	Standards	—
Existing applications	(6) Functions	Adapter, Proxy	(X)	—	(Implementation)	((4) Service list)	(2) Functions	(3)	(1) Function
Information Objects	(4) CRUD	—	X	(3) Data handling	Only for grouping	((2) State changes)	—	—	—
II. Service Refinement									
Single service in isolation	Validation by stakeholders	Coarse-grained	SType, Number of operations, right grained, aligned	SType (D, F, P, I); compensable business; coarse; idempotent; context-free	Low data transfer, not time critical, reusable, autonomous	Principles; SType (P, R, E); legal requirements	Business value, reusable, loose coupling	Reusable, right grained	Loose coupling, cohesion, reliability, business
Set of Services	Group operations by logical affinity	—	Transactions, data analysis, master data	Split by stability, functionality, type, object	Group operations by SType (task, entity)	Avoid functional overlap	Group functions (data, code)		Autonomy
III. Service specification									
	—	Small interface preferred, WSDL	4-6 operations per interface, WSDL	Small Interface	—	—	Structure, behavior, policy	Structure, behavior, policy; WSDL	WSDL, right granularity

Abbreviations: BU: Bottom-up; R: Reengineering; SType: Service type - D: Data, E: Entity, F: Function, I: Interaction P: Process, R: Rule, T: Task; TD: Top-down; W: Wrapping.

services. Moreover, the selection and combination of the patterns is left to the software developers. In all, pattern-based approaches are not suited as a guideline for SOA projects because they do not provide a design process from requirements to services.

Such design processes are prescribed by *hierarchical SOA design approaches* (e.g. [1], [2], [12], [13]) that proceed from some level of abstraction to a set of services. Common steps of these approaches are the *identification* of candidate services, their *refinement* and the *specification* of the refined services; see Table 1. The main differences consist in the direction and the sources of service identification: Services are

identified either *top-down* from business requirements or *bottom-up* from existing applications. *Hybrid approaches* combine both directions, though in the practical approaches usually one direction dominates. The sequences in which the sources of services are consulted are given by the numbers in Table 1; simultaneous checks have the same number. Altogether, top-down approaches dominate (see Table 1).

Top-down service identification usually starts from functional business areas [2], [13] or business processes [3], [13]. *Functional areas* are sets of related tasks referring to, e.g., departments or products; *business processes* additionally consider the order of tasks (*control flow*) as well as the roles performing the tasks. Mostly, candidate services are identified by hierarchically decomposing functional areas to atomic functions that correspond to (automated) activities in business processes. The control flow within the business processes is only used for the composition of services by BPEL engines (e.g., Oracle, IBM).

In some approaches, service identification relies on goals [1], [20] and information objects. *Goals* describe what should be achieved by a service [10]; they are either used to determine the functional scope of the SOA-based application [1] or to refine services and align them with business [2]. An *information object* is strongly interdependent information with similar life cycle that is important to business and can be the input or the output of activities in business processes. Information objects can be derived top-down from business requirements, domain ontologies or standards [1].

Bottom-up service identification usually starts from functions of existing application systems [8].

Service refinement involves the assessment of the identified service candidates by stakeholders. Criteria for refinement consider a *SOA service in isolation* (mostly its conformance to the SOA design principles) or in *relation* to other ones (e.g., grouping, splitting, avoiding overlap); see Table 1. Two additional criteria for refinement appear in Table 1: services types and granularity. *Services types* [12], [13] characterize the functionality provided by a service. Common service types are *entity* or *data* services (containing CRUD – create, retrieve, update, delete – operations with business semantics on information objects), *task* or *function* services (containing operations other than CRUD and meaningful to business) and *process* services (encapsulating the control flow between IT-supported activities; mostly the result of compositions). The term *granularity* is seldom defined (only in [20]). *Coarse-grained services*, which are a goal of refinement in several approaches [8], [2], alternatively mean business-relevant and aggregated functionality [8], [33], [20], a small number of operations with ‘comprehensive’ business scope [2] or the realization of business processes [9]. In contrast, *fine-grained services* correspond to a large number of interfaces and operations [6] or business objects [9]. So, implicit *measures of granularity* seem to be the ‘amount’ of business semantics of a service (large equals coarse-grained [24]) and the number of operations and interfaces (large equals fine-grained [6]).

Finally, *service specification* defines the service interface (operations and their signatures, given by message types for inbound and outbound messages). Existing practical SOA design approaches advocate small interfaces that group only ‘a few’ (four to six, [25]) operations.

Chronologically, academic approaches to design SOA lag behind the practical ones. Moreover, they often focus on service identification and, thus, only rudimentary discuss service refinement and specification. Solely the approaches [12] and [13] are

validated in a real industrial case; the validation of the other academic approaches consists in artificial samples. The industry-validated academic SOA design approaches [12] and [13] resemble the practical top-down ones.

Most of the approaches (see Table 1) are universal, some apply to distinct *development scenarios* only, namely [20]: *green field development*, where the SOA-based system doesn't exist yet; *reengineering*, where existing applications are newly implemented (or at least altered) based on SOA, or *wrapping* that leaves the existing applications unchanged and just adds an encapsulating SOA layer on their top.

4 Multiple-Case Study of Successful SOA Projects

4.1 Research Design

The challenges of SOA service design result from the complexity of real-world application situations, e.g., the required functionality and quality characteristics, heterogeneous IT infrastructures and the distribution of software units over business partners. The restricted validation of the academic SOA design approaches does not guarantee success in real-world settings. The validation of the practical approaches is doubtful as well – from a methodical point of view (done by the authors themselves) and from a strategic point of view (commercial interest in selling tools or consulting).

To get clarity about the factors that truly influence the design of the elementary services in service-oriented architectures, a multiple-case study was conducted. A *case study* is a qualitative empirical investigation of some contemporary phenomenon within its real-life context without the possibility to control the situation [32]. Here, five cases representing a variety of situations are analyzed. Generalizations of phenomena occurring in at least three distinct cases are assumed to be valid [31].

A *case* is a project where a large organization realizes SOA for some application software (or a landscape thereof) by implementing software services. Excluded are, thus, pure middleware or academic projects as well as projects where services are not yet implemented or only composed out of existing ones. To count as 'service-oriented', the architecture of a system or a system landscape must be in line with the definition of SOA derived in Section 2. Because of the outstanding importance of the WS-* standards (see Section 2), we require that WSDL specifications exist or can be generated for some of the service interfaces.

The *phenomena* to be observed within each case are the SOA services and the process of their design. We equate a *SOA service* with functionality, i.e., either operation or interface. A *SOA service design process* usually comprises steps in some chronology and design principles.

Each case has its *context*, which potentially influences service design, namely:

- *General context*: the company's branch, size, IT department and system landscape
- *Project context*: SOA motive, project scope, timing
- *Development context*: scenario (see Section 3), service provisioning (only within some controllable service inventory vs. to unknown consumers), and
- *Requirements*: functionality and quality.

The respective information for each case was gathered by a series of interviews (both face-to-face and by phone) with persons representing distinct roles (e.g., software

Table 2. Case context and SOA implementation results

	Case 1	Case 2	Case 3	Case 4	Case 5
Context					
Branch	Banking	Mail order business	Oil- and gas-production	Software industry	Swiss federal administration
Employees	46,700 in 40 countries	50,000 in 20 countries	30,000 in 40 countries	48,500 in 50 countries	Not investigatable 1 country
IT stuff	300	200	800	700	1750
System landscape	500 applications, mainframe (PL/1)	200 applications, central mainframe application (assembly language)	Custom-made applications (PL/SQL), packaged software, external IS	(irrelevant)	≈ 50,000 applications; technologically heterogeneous
SOA motivation	1. Reducing complexity 2. Integration 3. Flexibility 4. Testing use cases	1. Integration 2. Flexibility	1. Openness 2. Flexibility	1. Managing complexity 2. Flexibility	1. Reusability 2. Openness 3. Integration 4. Testing use cases
Scope	System landscape	Application (order processing)	Set of applications (oil and gas core)	Application (ERP for mid-sized companies)	System landscape
Start	1998/2002	2002	1997/2001	2002	2003
Scenario	Wrapping	Reengineering	Reengineering	Green field (& wrapping)	All
Implemented Services					
Consumer	Mostly known	Usually known	Partially unknown	Known and unknown	Mostly unknown
Number	650 operations	250 interfaces (50 infrastructure services)	80 interfaces (20 infrastructure services)	420 operations	> 70 interfaces
Operations per Interface	1	1... 9 (majority: 1; many 3)	3...5	1 ... 2	1 ... 22 (majority 1)
Versioning	Minor/major changes (3 concurrent versions)	Minor/major changes (2-3 times a year)	In preparation	New service in another namespace	Minor/major changes (1 time a year); 2 concurrent versions
Service Development and Deployment					
Style	CORBA; WSDL	WSDL	WSDL	WSDL	WSDL, REST
Platform	JavaEE, PL/1	JavaEE	JavaEE	Proprietary	.NET, JavaEE
Development Process	Waterfall	V Model, since 2009: Agile (Scrum)	Waterfall, since 2005: Agile (Scrum)	Iterative, derived from waterfall [11]	Waterfall

architects, software developers, project managers), by document analysis (design guidelines, service specifications, models) as well as by system analysis (interface descriptions, mostly in WSDL, of sample services). The final case report was checked for correctness by the contact persons of the companies. The following sections condense the most important case information.

4.2 Context of the Cases

The multiple-case study comprises five large organizations from distinct countries (Germany, Norway and Switzerland); see Table 2. In contrast to the Swiss federal administration, the four companies act world-wide. All organizations use several hundreds (thousands in the administration case) of heterogeneous applications. The *heterogeneity* of the applications refers to functionality, technology, life cycle phase, authorship (custom-made vs. packaged software) and controllability (in the companies or at business partners). Only applications at the business level were considered (e.g., enterprise resource planning, order management, customer data management, vehicle scheduling, residents' registration); neither desktop applications (such as word processing or spreadsheet software) nor Internet browsers or development tools.

Common *motives* to introduce SOA (see Table 2) were the *integration* of heterogeneous systems, *openness*, i.e., the capability to easily incorporate new service consumers (e.g., business partners), *flexibility* that enables quick reactions to new requirements by recomposing existing services, and *reuse*. From the clear cut of the SOA services the organizations expected reduced *complexity*. Finally, in two cases the capability to test isolated use cases (realized by services) was important.

All projects conceptually started around 2000; implementation mostly began around 2002/3. In three cases the whole system landscape should be based on SOA; the other cases focused on particular applications. All development scenarios are represented. In Case 4, program code from another ERP system was included (by wrapping) in the new SOA-based ERP system.

Focusing on an application usually dictates the services to be implemented from a functional point of view (service inventory [4]). However, if business partners are intended to use the services (SOA motive ‘openness’) or if the organization is split up in many independent sub-units, the service consumers are not known, and the services must be designed for unanticipated usage contexts. The corresponding service design processes are described in Section 4.3.

4.3 Service Design Processes

When the projects started in the five cases, practical SOA was in its infancy and no design guidelines were readily accessible. Each organization found its own process of designing services (see Table 3). In the Cases 1, 2 and 4, internal service design guidelines were created in the beginning of the projects, but only Case 1 and Case 4 had an accompanying *governance* process to review designed services.

Service design was always triggered by functional requirements stemming from the applications to be implemented (i.e., the necessary service inventory) or from ad-hoc requests. The rationale behind request-driven design was avoiding the (expensive) implementation of not ever consumed services. Quality criteria influencing service design were response time, secure data access and the ACID-properties (atomicity, consistency, isolation, durability) of transactions.

Roughly, the service design processes consist of four phases: (1) *Service identification* defines where to look for candidate services; (2) *initial design* states criteria the identified services have to keep; (3) *refinement* comprises checks that can trigger modifications of the initially designed services, and (4) *service specification* provides constraints or recommendations for the formal design in some definition language.

According to Section 3, potential sources of *service identification* are goals, functional areas, business processes, existing applications and information objects. The numbers in Table 3 reflect the sequences in which these sources were consulted; equal numbers for some case indicate concurrency.

All organizations analyzed *information objects* (called business entities, analysis classes or business objects; see Table 3) and their methods (operations) at the very beginning of service identification. Information objects such as ‘customer’, ‘order’ and ‘product’ exist in each case. Life-cycle operations (CRUD) of such information objects express distinct business semantics. For example, the candidate service ‘Update Vessel’ of Case 3 verifies that some vessel selected for an oil cargo meets all legal requirements [22]. Many business tasks can be easily transformed to CRUD operations, e.g., in the banking domain (Case 1) the activity ‘Buy Equities’ corresponds to the CRUD operation ‘Create Stock Exchange Order’.

Table 3. Service design processes of the cases

	Case 1	Case 2	Case 3	Case 4	Case 5
Design Trigger	Request	Inventory	Inventory, Request	Inventory	Request, Inventory
Quality criteria	Performance (response time), mass data transfer	Performance (response time), security	—	Transactionality	Security
Direction	Hybrid	Hybrid, bottom-up leads	Hybrid, top-down leads	Bottom-up	Mostly bottom-up
Governance	Yes	Design guidelines	In preparation	Yes	Partially
Service Identification					
<i>Goals</i>	—	—	—	—	(3) Capabilities
<i>Functional areas</i>	SOA Scope, service types	(1) Application (= business) functions	(1) Similar activities in functional business areas	Decomposition	—
<i>Processes</i>	(2) Sequences of activities	(atomic interaction sequences of a use case)	—	(2) A group of business object services	—
<i>Other</i>	(3) Business rules (4) Events	—	(3) Events (4) Business rules	—	(2) Events (Standards)
<i>Applications</i>	—	—	(2) Functions	—	(1) Interfaces
<i>Information Objects</i>	(1) Predefined operations on business entities; CRUD	(1) Methods of analysis classes	(1) CRUD	(1) Methods of business objects	(1) Information providing
Initial Design (Principles)	Abstraction; generalization (coarse-grained functions, big messages); business semantics; SType (D, P, BR); set-oriented services; test modus	Abstraction; smallest application (= business) function; statelessness; idempotence	Contract first' design; reuse data types; SType (E, T, I)	Business semantics; SType (A2A, B2B, A2X) + interaction patterns; modularity; context independence	Include flag .test'
Service Refinement	Similarity; formal correctness of interface; performance assessment	Same access rights; reusability; similarity; stability	Split: same information object + context; same business rules	Several steps of approval during bottom-up design [11]	Service market; minimal messages
Service Specification	Exactly one, preferably coarse-grained operation per interface; schema	—	Small interface, stable operations	Operations of the same interaction sequence	Small interfaces / operations / messages

Abbreviations: SType: Service type - D: Data, E: Entity, F: Function, I: Interaction P: Process, R: Rule, T: Task.

The second most important source of candidate services were atomic business functions (*tasks*), which can be gathered by

- Decomposing functional domains (Case 4) - in combination with bottom-up aggregation of business object services [11]
- Generalizing activities occurring in several business processes of some functional area (Case 3 [18])
- Grouping sequences of related activities (Case 1) or interactions (Case 2).

Thirdly, *events* and the operations to handle them were a source of candidate services. We observed the following types of *events*: state changes of information objects (e.g., Case 3: ‘Delivery completed’; Case 4: ‘Person married’; Case 1: ‘Data synchronization’), messages sent or received (e.g., Case 3: ‘Invoice sent’; Case 4: ‘Moving in from municipality’), timing (duration, time points – e.g., ‘New month’) and start/end of processes (Cases 1, 3).

Further sources of candidate services were the currently provided *interfaces* (Case 5) that hint at functional requirements of existing consumers, *business rules* and general *capabilities* that should be provided according to the organization’s goals (Case 5). Capability analysis ensures the completeness of a set of services, though it is usually bounded during refinement for pragmatic reasons (project efficiency).

During *initial service design*, SOA design principles such as abstraction and statelessness played a role. Moreover, services were designed to conform to some *service type*. Service types were defined from either a functional point of view (entity, task etc.) or from the interaction point of view, e.g., services between applications of one business partner (A2A), between systems at several business partners (B2B) or between an application and unknown consumers (A2X) [11].

Two cases explicitly provided a '*service test modus*' by either incorporating a specific 'test only flag' (Cases 1 and 5) or by designing a distinct 'validation service' if the test modus requires less input data (Case 1).

Only Case 1 considered service granularity: Initial service design aimed at *coarse-grained* services, which are associated with big messages and at the most two dependent entities as path length in service calls.

Reusability as well as *context* played a role in both initial service design and refinement: *Context independence* was an explicit design principle in Case 4. In Case 3, services were *split* during refinement to refer to the same information object in the same semantic *context*. For example, the information concept 'Cargo' has distinct interpretations depending on whether it is used in connection with terminal operations, e.g., storing at the port, or with supply operations, e.g., lifting, selecting a vessel. Hence, separate services must be defined.

Service refinement tries to increase the efficiency and effectiveness of the SOA project. *Efficiency* relates to the effort for service implementation, which was decreased by the following means: First, services that were derived during identification, but not demanded by some consumer (*service market*), were not implemented (Case 5). Secondly, if the functionality of a candidate service was already provided by an existing one (overlap $\geq 50\%$), the existing one was extended (*similarity*).

Effectiveness postulates that the service qualities are kept: Performance was assessed during refinement (Case 1). Security was guaranteed by splitting software services according to access rights (Case 2), by including only the necessary data in (small) messages (Case 5) or by using database functionality (Case 1). Reliable messaging and transactional data transfer were realized by *event bus* middleware in the Cases 1 and 5.

Service specification guides the design of the service interfaces: *Small interfaces* consisting of a few operations were preferred. Requiring just one operation per interface resulted from implementation (REST [7] in Case 5) or the wish to increase the stability and reusability of a service (Case 1). *Stability* of operations was a criterion for refinement (Case 2) or for specification (Case 3). The fact that several criteria appear in distinct phases demonstrates the independence of the service design approaches in the cases and the lack of a common guideline.

4.4 Implementation

Table 2 summarizes the results of the service design processes of Section 4.3 in terms of the current implementation. Only in Case 4 implementation is finalized.

All cases realized SOA by Web services described by WSDL (because of limited tool capabilities often WSDL 1.1) or REST [7]. REST was used in Case 5 for the GIS-application service (see Section 1), which just provides information without

transactional or security requirements and relies on a standard [15] that is naturally implemented by REST. Java Enterprise Edition (JavaEE) was the preferred implementation platform in the cases.

The tools for service deployment and integration are heterogeneous. *Event bus* middleware exists in the Cases 1 and 5; the Cases 2 and 4 rely on mainly message-oriented (EAI) middleware. The middleware of the Cases 1 and 4 also supports Remote Procedure Calls (RPC) – in Case 1 even within the CORBA component model. Message exchange via XML files is shared by the Cases 1, 3 and 5.

The total number of services per case ranges from currently close to one hundred to several hundreds. When we asked the companies for the number of ‘services’, they counted either (WSDL) interfaces or operations (see Table 2). Each interface comprises between one and around ten operations. Usually, the average number of operations per interface is low, i.e., many interfaces have one operation, the other ones two to three. The general design rationale behind small interfaces was stability; Case 5 additionally aimed at access protection. Huge interfaces mainly resulted from poor design in the ‘SOA learning phase’ (Case 5).

Service *versioning* is dealt with by the concept of minor and major changes [29]. *Minor changes* are backwards compatible, i.e., existing service consumers can remain unchanged while successfully using the new service versions. Examples of minor changes are the definition of new attributes or operations or the conversion of mandatory elements to optional ones. In contrast, deleting or renaming operations, changing data types of messages or semantic modifications of operations (even with unchanged syntax) are *major changes*, which are not backwards compatible and require the adaptation of the service consumers. The organizations usually bundle major changes and roll them out two to three times a year; mostly two to three active versions of some service are allowed at the same time.

In the Cases 2 and 3, the *development process* of the services is agile – to speed up development. In the other cases, service development follows the waterfall process, usually with at least one return from refinement to service identification if initial design criteria are violated. Several return possibilities lead to the overall iterative software service design process in Case 4 [10]. Because of the mandatory bidding process, any form of iterative or agile service development is excluded in Case 5.

4.5 Comparison of the Cases

All development scenarios (see Section 3) are represented. The SOA design approaches of the cases are hierarchical and identify services either hybrid (3 approaches) or bottom-up (2 approaches). To identify services, information objects are always checked first. Functional areas and aggregated activities are the second most important source, followed by events and business rules in alternating importance.

The reasons for preferring design that starts from information objects were distinct in each case:

- The organization of the company is not process-centric, but functional (Case 2).
- Information objects help in discovering similarities between business process models and, thus, keep the overall number of services small and their functionality generic (Case 3) [18].

- The business processes consuming the services are beyond the control of the organization designing the services (Case 5) [22].
- Service design by top-down decomposition of functional areas or business processes with dozens of variants [22] consumes too much time (Case 4); and the long analysis phase interferes with the principles of agile software development (Cases 2 and 3).
- Rank growth of ad-hoc services is avoided if design starts from information objects (Case 1 and Case 5).

Security is either incorporated in service refinement or handled outside the service (by middleware) or inside (the encapsulated application), respectively. Middleware assures reliable messaging and transactionality.

5 Conclusions

As a generalization of the successful SOA projects in the cases and the common ideas of the reviewed literature, the following recommendations concerning the design the elementary services for SOA can be given: The majority of approaches in the Tables 1 and 3 are hierarchical. Thus, a *hierarchical approach* should be used – because of the project structure it provides. The approach should consist of the (common) steps service identification, refinement and specification.

Service identification should aim at entity, task and infrastructure services. Service types, rarely used in the reviewed literature (see Table 1), were a common design principle in the cases as they give hints on where to look for services. *Entity services* are (business-driven) CRUD operations on information objects, i.e., the key terms of a domain that can be easily gathered from domain experts. *Task services* correspond to operations that are more complex than CRUD; they may involve more than one information object, represent interaction sequences or be guided by business rules. The cases relied on *aggregated* functions or (process) activities to identify task services, as opposed to detailed activities in the academic approaches. *Infrastructure services* are not business-driven, but needed for the systems to operate.

Refinement should check the services for reusability, service qualities (such as performance or security) as well as project efficiency (see Section 4.3). Project efficiency was largely ignored by the reviewed literature. Refinements are mostly realized by stakeholder reviews.

Service specification currently relies on the WS-* stack of specifications [23]. If services do not have requirements concerning transactionality and security and if appropriate tools exist, REST can be used (see Section 4.4). Service interfaces should group a small number (one to five) of operations, see Table 2. Small interfaces (e.g., 4-6 operations [25]) are also preferred by the practical approaches (see Table 1), not only with respect to granularity, but to isolate changes [19].

The validity of our contribution can be assumed for the following reasons: First, distinct methodologies (review, case study research) were applied. Secondly, the cases investigated are sufficiently disjoint in context, yet comparable – and vendor neutral (Case 4 refers to the internal SOA design approach of a software vendor).

Moreover, the proposed service identification is in line with more general criteria to decompose software into modules [19], namely data structure handling (entity services), sequences of functions (task services) and aids (infrastructure services).

References

1. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., Holley, K.: SOMA: A method for developing service-oriented solutions. *IBM Systems Journal* 47, 377–396 (2008)
2. Engels, G., Hess, A., Humm, U., Juwig, O., Lohmann, M., Richter, J.P., Voß, M., Willkomm, J.: *Quasar Enterprise: Service-oriented Design of Application Landscapes [Anwendungslandschaften serviceorientiert gestalten*]*. dpunkt, Heidelberg (2008) (in German only)
3. Erradi, A., Anand, S., Kulkarni, N.: SOAF: An Architectural Framework for Service Definition and Realization. In: *Proc. IEEE Int. Conf. on Service Oriented Computing (SCC 2006)*. IEEE, Los Alamitos (2006)
4. Erl, T.: *SOA Principles of Service Design*. Prentice Hall, Upper Saddle River (2008)
5. Erl, T.: *SOA Design Patterns*. Prentice Hall, Upper Saddle River (2008)
6. Feuerlicht, G.: Design of services interfaces for e-business applications using data normalization techniques. *Information Systems and e-Business Management* 3, 363–376 (2005)
7. Fielding, R.T.: *Architectural Styles and the Design of Network-based Software Architectures*. PhD Theses. University of California, Irvine (2000)
8. Fronckowiak, J.: *SOA Best Practices and Design Patterns. Keys to Successful Service-Oriented Architecture Implementation*. White paper, Oracle Corporation (2008), <http://www.oracle.com/technologies/soa/docs/soa-bp-design-patterns-whitepaper.pdf>
9. Heineman, G.T., Councill, W.T.: *Component-based Software Engineering – Putting the Pieces together*. Addison-Wesley, Boston (2001)
10. Kaabi, R.S., Souveyet, C., Rolland, C.: Eliciting service composition in a goal driven manner. In: Aiello, M., et al. (eds.) *Proc. of the Second Int. Conf. on Service Oriented Computing (ICSOC 2004)*, pp. 305–308. ACM Press, New York (2004)
11. Kätker, S., Patig, S.: Model-driven Development of Service-oriented Business Application Systems. In: *Proc. 9. Int. Conf. Wirtschaftsinformatik (WI 2009)*, vol. 1, pp. 171–180, Österreichische Computergesellschaft, Wien (2009)
12. Klose, K., Knackstedt, R., Beverungen, D.: Identification of Services - A Stakeholder-based Approach to SOA development and its application in the area of production planning. In: Österle, H., et al. (eds.) *Proc. of the 15th European Conf. on Information Systems (ECIS 2007)*, St. Gallen, pp. 1802–1814 (2007)
13. Kohlmann, F.: Service identification and design - A Hybrid approach in decomposed financial value chains. In: Reichert, M., Strecker, S., Turowski, K. (eds.) *Proc. of the 2nd Int. Workshop on Enterprise Modeling and Information Systems Architecture (EMISA 2007)*, pp. 205–218. Koellen-Verlag, Bonn (2007)
14. OASIS: *Reference Model for Service Oriented Architecture 1.0. Committee Specification* (2006)
15. OpenGIS: *OpenGIS® Web Map Server Implementation Specification, Version 1.3.0. Document Number OGC® 06-42* (2006)

16. The OpenGroup: SOA source book,
<http://www.opengroup.org/projects/soa-book/>
17. OpenSOA: Service Component Architecture: Assembly Model Specification, SCA Version 1.0 (2007)
18. Patig, S., Wesenberg, W.: Role of Process Modeling in Software Service Design. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC 2009. LNCS, vol. 5900, pp. 420–428. Springer, Heidelberg (2009)
19. Parnas, D.L.: On the Criteria To Be Used in Decomposing Systems into Module. *Communications of the ACM* 15, 1053–1058 (1992)
20. Papazoglou, M.P., van den Heuvel, W.-J.: Service-oriented design and development methodology. *Int. Journal of Web Engineering and Technology* 2, 412–442 (2006)
21. Papazoglou, M.P., van den Heuvel, W.-J.: Service oriented architectures: approaches, technologies, research issues. *The VLDB Journal* 16, 389–415 (2007)
22. Patig, S., Müller, W.: Event-driven Design of SOA for the Swiss Citizen Registration [Event-Driven-Design serviceorientierter Architektur für das schweizerische Personermeldewesen*]. In: Proc. Vernetzte IT für einen effektiven Staat Fachtagung Verwaltungsinformatik (FTVI). LNI, vol. 162, pp. 183–194. Gesellschaft für Informatik, Bonn (2010) (in German only)
23. Pautasso, C., Zimmermann, O., Leymann, F.: RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. In: Proc. 17th international Conf. on the World Wide Web (WWW 2008), pp. 805–814 (2008)
24. Sametinger, J.: *Software Engineering with Reusable Components*. Springer, Berlin (1997)
25. SAP AG: *Enterprise Services Design Guide* (2009),
http://www.sap.com/platform/netweaver/pdf/BWP_ES_Design_Guide.pdf
26. Schulte, R.W., Natis, Y.V.: *Service Oriented Architectures, Part 1*. Research Paper, ID Number SPA-401-068, Gartner (1996)
27. Szyperski, C.: *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, Harlow (1996)
28. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.F.: *Web Services Platform Architecture – SOAP, WSDL, WS-Policy, WS-Addressing, WS_BPEL, WS-Reliable Messaging, and More*. Prentice Hall, Upper Saddle River (2005)
29. Wright, M., Reynolds, A.: *Oracle SOA Suite Developer's Guide*. PACKT, Birmingham/Mumbai (2009)
30. W3C: *Web Services Description Language (WSDL) 2.0, Part I: Core Language*. W3C Note (2007)
31. Yin, R.K.: The Case Study as a Serious Research Strategy. *Knowledge: Creation, Diffusion, Utilization* 3, 97–114 (1981)
32. Yin, R.K.: *Case Study Research: design and Methods*, 4th edn. SAGE Publications, Thousand Oaks (2004)
33. Zhang, Z., Liu, R., Yang, H.: Service Identification and Packaging in Service Oriented Re-engineering. In: Proc. of the 17th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE 2005), Skokie, pp. 620–625 (2005)

Part III

Distributed Systems

Division of Water Supply Systems into District Metered Areas Using a Multi-agent Based Approach

Joaquín Izquierdo, Manuel Herrera, Idel Montalvo, and Rafael Pérez-García

Fluing-IMM, Universidad Politécnica de Valencia, Cno. de Vera, s/n, 46022 Valencia, Spain
{jizquier,mahefe,imontalvo,rperez}@upv.es

Abstract. Technical management of large water supply systems (WSS) is an increasingly complex problem. Water companies managing these systems have witnessed how the mathematical models of their networks lose accuracy and their engineering tools become obsolete. Consequently, they have no clear vision of the balance between production and distribution, that is to say, between supply and demand. As a result, water companies are interested in improving the control and management of their networks. One of the methods attracting great interest is that of division into DMAs (district metered areas). Division into DMAs splits an interconnected and intricate network into smaller, virtually independent sub-networks that can be better managed. However, the complexity of the problem of creating DMAs demands efficient techniques. In this contribution we use a multi-agent based approach that takes advantage of the distributed nature of WSS.

Keywords: Water supply management, Distributed systems, District metered area, Multi-agent systems.

1 Introduction

During the past fifteen years, the Multi-agent Systems (MAS) paradigm has been applied in a large number of fields including not only multimedia and computer entertainment, virtual reality, web-based interfaces, and tutoring systems. Moreover, as MAS technology is now mature, it is being applied in new fields where design and implementation of distributed intelligent systems could be useful. MAS modelers can handle different levels of representation within a unified conceptual framework – ranging from simple individuals to very complex groups. Such versatility makes MAS especially suitable for simulating complex systems. MAS are used in an increasing number of scientific domains: sociology, biology, physics, chemistry, ecology, economy, etc. [1].

In the water field, the trend in recent years has been to include multi-agent techniques as an interesting alternative for solving complex problems. See, for example, [2] on multi-agent applications in urban hydraulics; [3] on control systems for municipal water; [4] on water pollution diagnosis; [5] on water demand management for a free access water table; [6] on water quality; [7] on water management at catchment scale; [8] on water management at river basin scale; [9] on allocation of scarce water; [10] combining multiobjective optimization with MAS in a

new paradigm called agent swarm optimization (ASO); [11] for visualizing plastic pipes in WSS using GPR (ground penetrating radar) images; [12] for simulation of hydraulic transients.

However, urban water supply management, thoroughly plagued with complexity, has received little impact from MAS so far. Some examples include [13] using agents to control the physical equipment of a water supply; [14] following this research line and examining a criterion for maintaining continuity and reliability in water supply; [15] developing a modified multi-agent genetic algorithm to optimize water-using networks; and [16] building a suitable environment to simulate DMA partitions in a WSS by using the multi-agent metaphor. This last work may be considered as an antecedent of the current contribution in relation to the methodology and approached water problem.

1.1 Water Supply Complexities

Water supply is one of the more recognizable and important public services contributing to quality of life. It exhibits a number of characteristics that are quite different from those of other public services. Distribution is irregular, both in temporal and spatial terms. In addition, operation can be analyzed from very different perspectives. As water is for consumption, aspects related to health need to be considered: water quality and the appropriate control measures to maintain quality during residence time in the network. However, water systems suffer a number of operational and environmental conditions, which lead to progressive and insidious deterioration. There are a variety of factors involved: loss of pressure, due to increasing inner roughness of pipes; breakage or cracking of pipes, caused by corrosion and mechanical and thermal charges; and loss of water (leaks), due to pipe breaks and cracks, with their corresponding economic loss, and third party damage. All these factors can create a risk of contamination. Because of the complexity of water systems, mainly due to the interconnected nature of sources and consumption points, it is extremely difficult to balance production and distribution, that is to say, to control the water supplied and consumed. Division of the network into DMAs follows a divide-and-conquer strategy that splits a large and highly interconnected distribution network into smaller and virtually independent networks – each supplied by a pre-fixed number of sources.

Independence can be physically achieved in a number of ways: by closing valves in existing pipes, by sectioning existing pipes, by introducing new pipes that redistribute the flow, etc.

Manageable DMAs will enable action to be taken to improve the control and management of such important aspects of water distribution as water quality and the intensity and spatial and temporal distribution of leaks. DMAs will help reduce unaccounted-for water loss and improve the water-tightness of the system – thereby saving huge amounts of water and preserving water quality. Substantial water savings can be forecasted if sectorization is enforced systematically.

1.2 Division of WSS into District Metered Areas

Real water distribution systems may consist of thousands of consumption nodes interconnected by thousands of lines, as well as the necessary elements to feed the

network. These networks are not usually the result of a unique process of design, but the consequence of years of anarchic response to continually rising demands. As a consequence, their layouts lack a clear structure from a topological point of view. This fact renders these networks difficult to understand, control, and manage. In the case of small networks, simple techniques, sometimes of a visual character, enable division into a few DMAs. But this task is unthinkable for very large networks because their complexity renders the problem virtually unfeasible. As a consequence, new algorithmic capabilities, not implicitly contained in the hydraulic model, would be of great interest.

The main objective of creating DMAs (also called sectorization) is to obtain the distributed and manageably scaled information necessary to perform key actions in each sector [17]. These actions include:

- Audit the hydraulic efficiency or NRW (non-revenue water),
- Characterize the demand curve, especially the night flow,
- Quickly detect leaks by analyzing the evolution of the night minimum flow,
- Check the results of search campaigns and repair leaks quickly,
- Detect fraud, under-registration, or diverse errors of measurement,
- Reduce maintenance costs,
- Plan investments when guiding supply to the sectors with more NRW.

Working on a sector reduces the system inspection area. It thereby facilitates the detection, identification and monitoring of possible abnormal states of water supply [18]. Other improved water management topic is about predictive models. These are more accurate in a DMA than in the whole network, avoiding biases derived from achieving forecasts in smaller areas than DMA [19]. Other promising lines are the optimization of DMAs pumping scheduling, and the problem of sensor location detecting pathogen intrusion in real-time.

The procedure to define the hydraulic sectors implies [20]:

1. Obtaining the number of independent sectors in a network layout. A sector of the network is said to be independent when it is supplied exclusively from its own water sources, and is not connected to other sectors in the network.
2. Obtaining the set of network nodes belonging to each individual sector.
3. Revising proposed sectorization actions, such as valve closing or pipe sectioning, in case such actions may cut the water supply for some parts of the network.
4. Defining the area served by each water source, and the contribution of each source to the consumption of each network node.

The first and third of these four tasks are crucial for detecting errors in the network layout and in proposed sectorization decisions. The second task is essential for water audits, and the fourth task is important for defining and visualizing any proposed sectorization.

A district metered area is a part of the water distribution network that is hydraulically isolated, temporally or permanently, and ideally has just one supply node equipped with a flow meter. DMAs are small zones of the system and usually contain between 500 and 3000 service connections.

The concept of DMA management was first introduced to the UK water industry in the early 1980s [21], and it has been used as an instrument to monitor and reduce the

level of leaks in water supply systems. The technique was mainly developed in Europe, and has been used in Latin America since the 1990s, while it is less often used in the United States and Canada. The development of DMAs has been strongly empirical, being based on technical experience and with very few scientific contributions. It is necessary to highlight the contributions in UKWIR [22] and IWA [23]. Recently, some proposals have been presented for a conceptual and scientific framework – such as [24] relative to the periodic acoustic surveys in a DMA; or [20] in applying graph theory to establish the division of DMAs.

In this contribution, we explore the division of a water supply system into DMAs by using a multi-agent approach. Complex problems, such as the problem considered in this article, can be resolved using distributed agents because the agents can handle combinatorial complexity in a real-time suboptimal approach [25].

The structure of this work is as follows. Firstly, we introduce the agent-based ingredients, then describe the used implementation, and finally, present the main results. A conclusions section closes the work.

2 Multi-agent Metaphor

In the study of complex systems, computer programs have played an important role. However, the actual process of writing software is a complicated technical task with much room for error. The multi-agent philosophy adopts a modeling formalism based on a collection of independent agents interacting through discrete events. Simulation of discrete interactions between agents stands in contrast to continuous system simulations, where simulated phenomena are quantities in a system of coupled equations.

An agent is any actor in a system: any entity that can generate events that affect itself and other agents. In the problem we consider here, agents are consumption nodes, connecting pipes, supply sources, ground and underground patches containing the network; as well as district metered areas, which are sets of nodes, pipes, sources, and patches. Even the whole network is an agent following specific scheduled actions. In these last two cases, the behavior of an agent is defined by the emergent actions of the agents it contains.

Agents define the basic objects in the system – the simulated components. The simulation occurs in the modeled world itself, and it is frequent to speak of agents as living in an environment, which, as said, can be an agent itself.

Agents are one of the abstractions that have been most frequently used to describe and implement proactive intercommunication among modules in intelligent distributed systems. There are some properties which agents should satisfy [26]: reactivity, perceiving their environment; pro-activeness, being able to take initiative; and social ability, interacting with other agents. The MAS paradigm proposes a scenario where independent, goal-directed, and environment-aware units (the agents) become coordinated (by collaborating or competing) to accomplish complex tasks [27]. Some advantages of MAS are a solid conceptual grounding (they can be depicted using well-defined abstract entities and operations), encapsulation of their components (which hides agent policies and promotes scalability), communication facilities (high-level protocols are used), and parallel execution (resulting in better performance and robustness) [28].

Once agents have been defined and their relationships established, a schedule of discrete events for these objects defines a process occurring over time. Individual actions take place at some specific time; and time only advances alongside events scheduled at successive times. A schedule is a data structure that combines actions in the specific order in which they should execute. The passage of time is modeled by the execution of the events in some sequence. Instructions are given to hundreds or thousands of independently operating agents. This makes it possible to explore the connection between the micro-level behavior of individuals; and the macro-level patterns that emerge from the interaction of many individuals.

A final step consists in observing the model and recording what is happening. Observers perform these actions. In most platforms there are also agents with specific tasks, such as plotting, storing data, monitoring and displaying certain variables, etc.

Agents should possess the following properties: autonomy, mobility, reactivity, pro-activity, adaptability, communicativeness, robustness, learning ability, task-based orientation, and goal-based orientation [29].

3 Implementation

NetLogo [30] is an environment for developing complex, multi-agent models that evolve over time. It is possible to create populations of changing agents in a suitable grid of stable agents. The evolution of agents can take different forms. Agents can be created, move, change their properties, change their behavior, change their nature or breed, and even die.

Our model is created from GIS data defining the physical and topological network characteristics. The experimental data were obtained from GIS models of two real moderately-sized networks that have been studied by the authors within a joint research project with an international water company. These networks are parts of two water distribution systems in two Latin American cities.

The area is divided into squares (patches), which gives some raster format to the environment. Patches represent the ground (underground) where pipes and nodes are buried. Figure 1 shows a section of one network. Patches are used to define areas occupied by the different divisions that will be created. Consumption nodes (small circles) are agents (turtles) of a certain breed with a number of associated variables. Among the user-defined variables, elevation and demand are the most important. During the process, color is used to define the DMA that the agents belong to. Pipes (lines) are links (in the problem under consideration they are undirected links). Each pipe connects two different nodes and also has some associated variables. The main user-defined variables are diameter and length. Source points (squares) are another breed of turtles, whose variables are the average of the demand they supply and the DMAs they feed. Patches, sources, nodes, and pipes are spatially fixed agents in the sense that, obviously, they do not change their position with time. Instead, they change their properties, especially color, and as a result they eventually belong to one DMA or another. Initially, sources, nodes, and pipes are presented in light grey, since no district structure is available at the setup. In this model, the user decides the number of DMAs to be built. Then, randomly the same number of source points are selected to be the seeds for the corresponding districts.

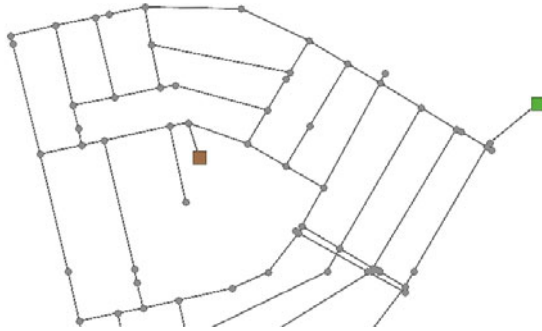


Fig. 1. Detail of a network in NetLogo environment

Two consecutive processes are launched that perform the sought division into DMAs. First, a process of clustering by exploration provides an initial division; then a fine-tuning process of negotiation of boundaries optimizes the division.

3.1 Clustering by Exploration

Upon setup, these turtles start a process of probing their neighboring nodes and checking the likelihood of their neighbors being assimilated into the same would-be DMA as themselves. This likelihood is derived from a number of tests, which are performed on the basis of sources, nodes, and pipe properties:

- The total length of the current DMA must be bounded by minimum and maximum values for the total length of the set of its members,
- The elevation of a new candidate must be in a certain range around the average elevation of the current DMA,
- The total demand of the sector must be between certain pre-fixed limits,
- The associated sources must be able to meet that demand,
- The geometrical properties of the area occupied by a DMA must exhibit certain basic requirements (connectivity, convexity, etc.),
- Other properties.

Nodes and pipes passing these tests are assimilated to the winning DMA, and the process is repeated again. To this purpose, neighboring nodes for every hydraulic sector or division are explored in each step of the algorithm. These nodes are given a certain probability of belonging to a given district. This probability reflects the difference between the elevation of the node and the average elevation of the district; and the difference between its demand and the average demand of the sector. In this way, simple greedy competition based on minimum distance among the districts adds increased probabilistic richness to the process. As a result, the model agents, performing a mixture of individual and collective actions, can explore good network sectorization layouts by trying to meet the equation

$$\text{Min} \sum_{c=1}^C [\alpha_z (z_i - \bar{z}_c) + \alpha_d (d_i - \bar{d}_c)], \quad (1)$$

where α_z and α_d are the associated weights to level, z , and demand, d , to carry out the clustering configuration; \bar{z}_c and \bar{d}_c are the level and demand averages in cluster c , respectively. C is the total number of different DMAs or clusters.

Decisions performed this way achieve homogeneity in the proposed DMAs. Table 1 present a pseudo-code for this algorithm.

Table 1. MAS-clustering algorithm based on homogeneity of the districts

Global:	0. Initial settings
Global:	1. Select source nodes: assign different groups to them
Agent:	2. Scan cluster neighboring: 2.1 Select a random node from their neighborhood 2.2 Is it on my cluster? Yes: Go to 2.1 No: Continue
	3. Check selected neighbor: 3.1 Is difference on demands and levels $< Limit1$? Yes: Add to cluster and Go to 2 No: Go to 2.1

Once a new configuration has been built, other requisites (sector size, connectivity, etc.) are checked before validating the configuration. Borders between sectors – which can be fuzzy – are identified by specific pipes, showing the location of cut-off valves, which are used to isolate the sectors during operation.

The natural consequence of this process is that different DMAs are built and minimal sets of sectioning lines are identified. Nevertheless, some nodes may end up disconnected and borders between sectors may be poorly defined. Even though the main objective is the identification of DMAs and cut-off lines, the information about disconnected nodes and overlaps between nodes and pipes of different sectors can also be used by the network manager. These circumstances, which show that the desired balance is still undergoing some debate regarding assignments, can be used to detect errors in network data, propose candidate areas for sensitivity analyses, and encourage various actions aimed at improving the layout and/or the topology of the network. The next paragraph describes an interesting sensitivity analysis based on energy efficiency.

3.2 Negotiation of Boundaries

Most of the points furthest from water sources are near boundaries between districts. These points are usually the worst represented by the clusters to which they belong. We propose adding an energy criterion after performing the first division to re-assign these points in an efficient way. So far, we have established sectors based on level and demand distances. Now, we improve this dissimilarity measure by taking into account

the length of the paths from source points to the boundaries and changing the average district level to the level of source points. A sensitivity analysis is thereby proposed that negotiates the boundaries previously established from an energy point of view. This process continues following our previous main target of homogeneity and minimal number of used valves. As a result, the global behavior of the model agents, which perform a mixture of individual and collective actions, can explore the best network layout that provides efficient supply.

To this end, nodes close to the boundaries are activated to negotiate their cluster membership. The new paradigm re-classifies these points to a different membership if the cost associated with supplying them is sufficiently large. Now, the general agent-classification process derives from equation (2):

$$Min \left\{ \lambda_{\beta} \sum_{c=1}^C [\beta_l l_{i,c} + \beta_z (z_i - \bar{z}_c)] + \lambda_{\alpha} \sum_{c=1}^C [\alpha_z (z_i - \bar{z}_c) + \alpha_d (d_i - \bar{d}_c)] \right\}, \quad (2)$$

where β_l is the associated weight to distance in coordinates, $l_{i,c}$, between the checking node and the source point of district c ; β_z is the weight of the difference between levels in the supply. λ_{β} is the importance of negotiating the energy effort of the hydraulic system and λ_{α} weights the above equation (1) by homogenizing DMA demands and levels.

The second phase of the algorithm takes into account the energy costs associated with supply (see Table 2). Working with the same agents will provide continuity for the process. Nevertheless, these agents will change their point of view with respect to the clustering problem and will change certain memberships if the distances from the sources to consumption nodes are large; so following the idea behind equation (2).

Table 2. Negotiating cluster boundaries

Global:	0. Set the boundaries of the clustering solution
Global:	1. Select random nodes: verifying their cluster membership
Agent:	2. Scan neighboring cluster: <ul style="list-style-type: none"> 2.1 Select a random node from neighborhood 2.2 Is it on my cluster? <ul style="list-style-type: none"> Yes: Go to 2 No: Continue
	3. Check selected neighbor: <ul style="list-style-type: none"> 3.1 Is distance to the proposed new water source < current distance to source? <ul style="list-style-type: none"> Yes: Is difference on demands and levels < <i>Limit</i>2? Re-assign to cluster and go to 2 No: Go to 2.1

Two constraints, namely *Limit1* and *Limit2*, are used in these algorithms. Both are related with homogeneity conditions to establish suitable clusters in the sense of becoming efficient DMAs. *Limit1* specifies the maximum allowed difference between node levels and demands, and identical variables (on average) for each cluster. This gives support to agents making decisions about whether to add a node to a cluster. However, this decision is poorly achieved in the cluster boundaries. *Limit2* enables the negotiation of each membership for those areas, relaxing the homogeneity constraints of the first algorithm and taking into account energy gains in the supply.

3.3 Practical Issues for Implementation

Through the use of additional interface elements, the user can manage the course of the simulation by changing various parameters (Figure 2 shows the interface for one of the studied networks).

The membership probability measurements of a node with respect to a district depend on elevation and demand, and can be modified by using the slider labeled 'weight-demand'. The user can also decide a priori the number of hydraulic sectors to be built by selecting an option from the chooser labeled 'n-clus'. By using the switch labeled 'zoning' the user can also ask the program to color the patches occupied by the different hydraulic sectors. This option, as well as offering an interesting visual value, enables the user to decide if the districts displayed exhibit good topological properties. Certain convexity and/or compactness properties are desirable for districts. By default (option off) the different colors for the pipes and nodes make clear the division of the hydraulic network into districts. By flipping the switch to 'on', patches are colored according to the color of the nodes and pipes they contain. This option is useful for revealing overlaps between sectors which, as explained before, can be used to produce suitable sensitivity analyses.

The simulation results can be visualized on screens, plots, and graphs; and data can be stored for further processing in hydraulic simulation software and for decision-making support.

Figure 2 also presents several displays showing some of the used parameters, such as the average elevation of the different sectors and the number of pipes in the sectors. Of special importance is the display labeled 'n-valves' which shows the number of sectioning links connecting different sectors; these are pipes that enable isolation-communication between two sectors and provide the engineer with useful information about the candidate pipes for sectioning and where to install cut-off valves to isolate the districts. Engineers must make important decisions about the need to install closing valves in existing pipes, and about sectioning those pipes, and/or introducing new pipes that redistribute the flow in more a reasonable manner.

The process is able to find good solutions for the connectivity between DMAs. As a consequence, the number and location of the closure valves is optimized for a given layout. In addition, nodes are assigned to sectors in a remarkably stable way that further stabilizes during the evolution of the process (see the demand plot on Figure 2). In addition, the best partitions can be found with more frequency during different runs of the process. As a result, by repeating the process a certain number of times, the engineer can make a final decision that may, or may not, coincide with any of the

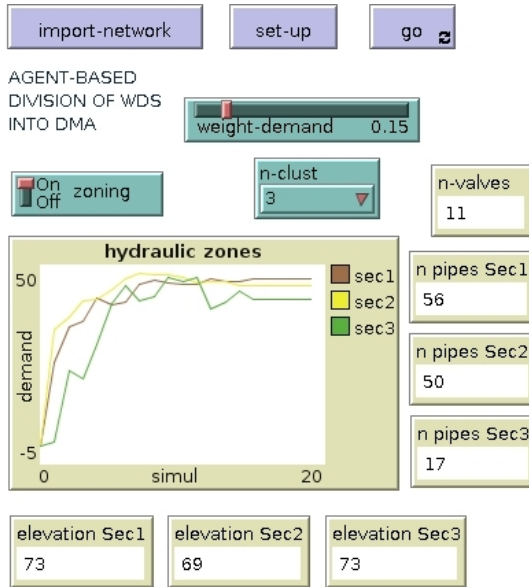


Fig. 2. Interface including parameter selectors and monitors

obtained partitions – these being used as a basis for the decision. Our simulation model helps managers communicate with domain experts, because they can perform their analyses using solved modeled situations.

4 Results

To show the performance of the described process we present for the sake of brevity, only the results corresponding to one of the studied networks (see Figure 3) fed by six reservoirs and made from 479 lines and 333 consumption nodes; total length being 48 km, and total consumed flowrate being 91 liters per seconds.

After 20 runs of the model simulating the partition of this network into hydraulic sectors, the configuration shown in Figure 4 was obtained in 80% of the cases. As a result, three sectors are obtained and these are isolated with 34 cut-off valves.

These sectors have 118, 199, and 162 pipes, respectively. The average elevations for these sectors are 155m for the upper-right sector, 157m for the left sector, and again 155m for the lower-right sector.

All of these sectors satisfy the requirements for becoming valid hydraulic sectors in terms of maximum and minimum total pipe lengths.

Finally, it is noteworthy that the validity of the district division has been checked using EPANET2 [31]. The analyses performed show that the proposed division effectively cuts the water supply for the desired parts of the network. Also, the entire network and individual sectors maintain all the design requirements. As a consequence, the proposed division into DMAs is perfectly feasible and reliable.

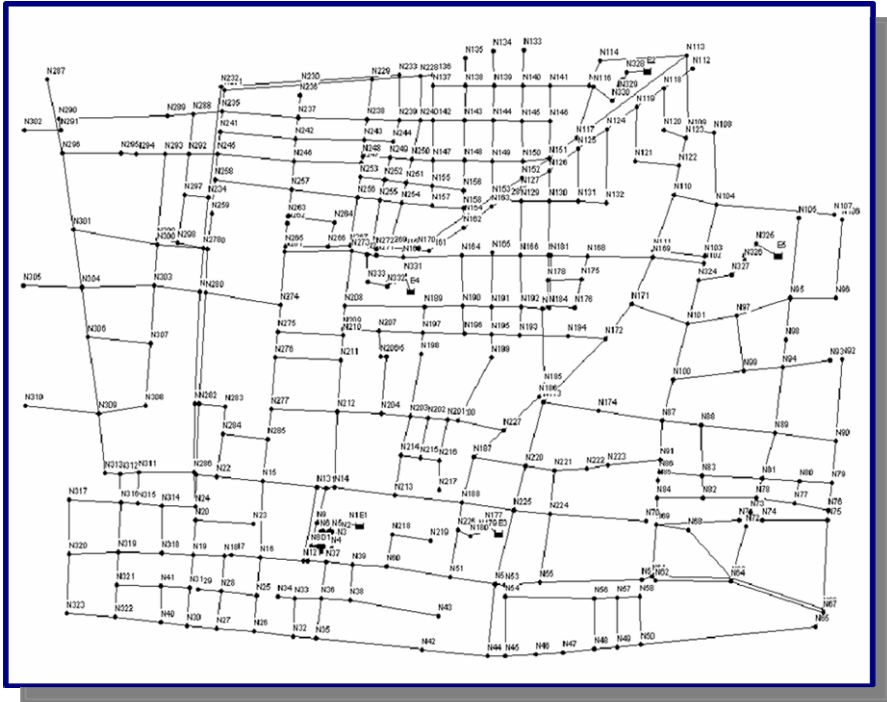


Fig. 3. Layout of the studied network

5 Conclusions

The multi-agent metaphor has been used with success in various areas and is reasonably applicable in the water management field. In addition to the traditional centralized architecture of a single reasoning agent (the computing counterpart of human decision support), it is possible to use systems of reasoning agents, or to apply multi-agent simulations to verify hypotheses about various processes in water distribution. Partial implementations of multi-agent applications are expected to simplify communication with domain experts during the process of modeling expert knowledge, identifying needs, and summarizing requirements for the final application. Among the various scenarios using multi-agent systems in the scope of decision support for a water management company, we have focused on the division of a network into district metered areas.

The inclusion of agents with negotiation abilities proposes a sensitivity analysis of the solution previously found via agent cooperation and competition. However, in addition, it is the start line for multi-objective approaches to be developed in the future.

The model may be applied to larger networks. Indeed, taking into account that the considered network in Figure 3 is medium-sized and running times are slow (ranging between 10 and 20 seconds on a PC with an Intel Core 2 Duo T5500 1.66 GHz processor for the case considered), no added difficulties are foreseen.

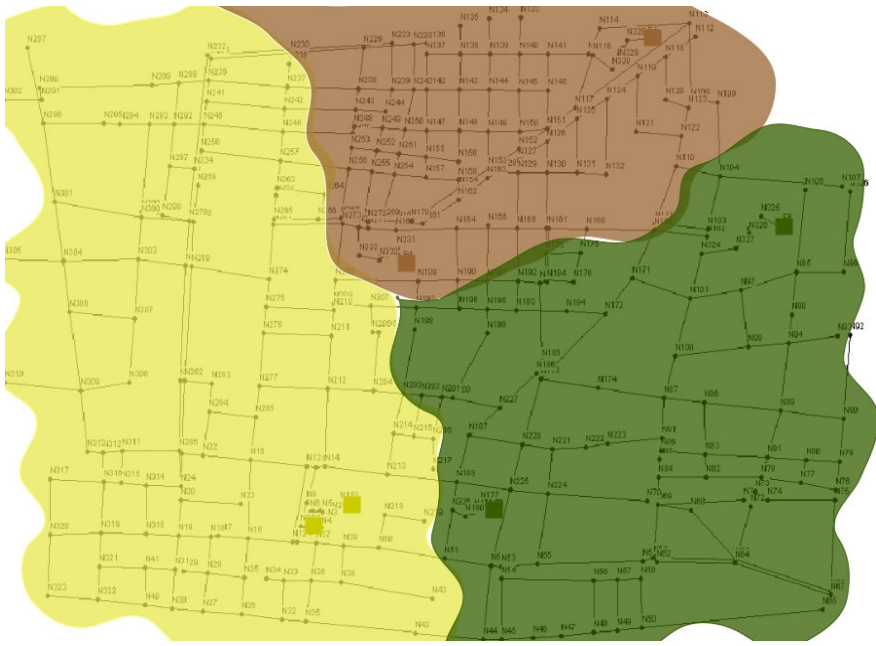


Fig. 4. Final distribution of hydraulic sectors for one of the studied networks

Division into DMAs helps the decision-making process, as well as the implementation of suitable actions to improve the control of a network and give solutions for important aspects of water distribution management – such as leaks and water quality.

Various lines of research may be followed in the future. One line of action will be addressed by exploiting the presented model in a number of ways: adding new conditions to cluster building; refining the implemented clusters following new criteria; and generally improving the performance of the model. An interesting improvement would consider starting the process automatically, instead of requiring the user to define a priori the number of sectors, and so make a division into an optimum number of sectors. Another line of research will focus on the development of other scenarios for multi-agent applications in the water supply field, including aspects related to water quality and other management issues.

Acknowledgements. This work has received the support of the project IDAWAS, DPI2009-11591, of the Dirección General de Investigación of the Ministerio de Ciencia e Innovación of Spain, including FEDER funds, and the action ACOMP/2010/146 of the Generalitat Valenciana. The use of English in this paper was revised by John Rawlins.

References

1. Drogoul, A., Vanbergue, D., Meurisse, T.: Multiagent based simulation: Where are the agents? In: Sichman, J.S., Bousquet, F., Davidsson, P. (eds.) MABS 2002. LNCS (LNAI), vol. 2581, pp. 1–15. Springer, Heidelberg (2003)
2. Izquierdo, J., Montalvo, I., Pérez, R.: Aplicaciones de la inteligencia colectiva (multiagente) para la optimización de procesos en hidráulica urbana. VIII Seminario Iberoamericano – SEREA Influencia sobre el Cambio Climático, la eficiencia energética, de operaciones y Sistemas de Seguridad en el abastecimiento y el drenaje urbano (2008)
3. Kotina, R., Maturana, F.P., Carnahan, D.: Multi-agent control system for a municipal water system. In: Proceedings of the 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases, Madrid, Spain, pp. 464–469 (2006)
4. Nichita, C., Oprea, M.: Water Pollution Diagnosis with a Multi-Agent Approach. In: Proceedings of Artificial Intelligence and Soft Computing (2007)
5. Feuillette, S., Bousquet, F., Le Goulven, P.: SINUSE: a multi-agent model to negotiate water demand management on a free access water table. *Environmental Modelling and Software* 18(5), 413–427 (2003)
6. Hai-bo, L., Guo-chang, G., Jing, S., Yan, F.: Multi-agent immune recognition of water mine model. *Journal of Marine Science and Application* 4(2), 44–49 (2005)
7. Becu, N., Perez, P., Walker, A., Barreteau, O.: CatchScape: An integrated multi-agent model for simulating water management at the catchment scale, a northern Thailand case study. In: Ghassemi, F., et al. (eds.) Integrating Models for Natural Resources Management Across Disciplines, Issues and Scales. International Congress on Modelling and Simulation, Canberra, Australia, pp. 1141–1146 (2001)
8. Mikulecký, P., Bodnárová, A., Olševičová, K., Ponce, D., Haviger, J.: Application of multi-agent systems and ambient intelligence approaches in water management. In: 13th IWRA World Water Congress, Montpellier, France (2008)
9. Hailu, A., Thoyer, S.: Multi-Unit Auctions to Allocate Water Scarcity Simulating Bidding Behaviour with Agent Based Models. LAMETA Working paper 2005-01, EconWPA (2005)
10. Montalvo, I., Martínez, J. B., Izquierdo, J., Pérez-García, R.: Water Distribution System Design using Agent Swarm Optimization. In: Water Distribution System Analysis 2010 – WDSA 2010, Tucson, AZ, USA (2010)
11. Ayala-Cabrera, D., Herrera, M., Izquierdo, J., Pérez-García, R.: Towards the visualization of water supply system components with GPR images. In: Mathematical Models of Addictive Behaviour, Medicine & Engineering, UPV, Valencia, Spain (2010)
12. Izquierdo, J., Montalvo, I., Pérez-García, R., Izquierdo, F.J.: Hydraulic Transient Simulation in Networks using a Multi-agent based approach. In: Water Distribution System Analysis 2010 – WDSA 2010, Tucson, AZ, USA, September 12-15 (2010)
13. Gianetti, L., Maturana, F.P., Discenzo, F.M.: On the Computational Geometry of Pocket Machining. LNCS, pp. 500–510. Springer, Heidelberg (2005)
14. Maturana, F.P., Kotina, R., Staron, R., Tichý, P., Vrba, P.: Agent-based water/waste water control system architecture. In: IADIS International Conference Applied Computing (2006)
15. Cao, K., Feng, X., Ma, H.: Pinch multi-agent genetic algorithm for optimizing water-using networks. *Computers & Chemical Engineering* 31(12), 1565–1575 (2007)
16. Izquierdo, J., Herrera, M., Montalvo, I., Pérez-García, R.: Agent-based division of water distribution systems into District Metered Areas. In: International Conference on Software Data Technologies, Sofia, pp. 83–90 (2009)

17. Aguas de Valencia: Sectorización, <https://www.aguasdevalencia.es/portal/web/Tecnologia/Tecnologia/GestionRedes/Sectorizacion.html> (accessed: 24 September 2010)
18. Herrera, M., Pérez-García, R., Izquierdo, J., Montalvo, I.: Scrutinizing changes in water demand behavior. In: *Positive Systems Lecture Notes in Control and Information Sciences*, pp. 305–313. Springer, Heidelberg (2009)
19. Herrera, M., Torgo, L., Izquierdo, J., Pérez-García, R.: Predictive models for forecasting hourly water demand. *Journal of Hydrology* 387, 141–150 (2010)
20. Tzatchkov, V.G., Alcocer-Yamanaka, V.H., Bourguett-Ortíz, V.: Graph Theory Based Algorithms for Water Distribution Network Sectorization Projects. In: *8th Annual Water Distribution Systems Analysis Symposium*, Cincinnati, Ohio, USA, August 27-30 (2006)
21. Morrison, J.: Managing leakage by District Metered Areas: a practical approach. *Water* 21, 44–46 (2004)
22. UK Water Industry Research Ltd: *A Manual of DMA Practice*. UK Water Industry Research, London (1999)
23. IWA Water Loss Task Force: *District Metered Areas: Guidance Notes*. Version, p. 100 (February 2007), http://www.waterlinks.org/upload_file/8f4be72e-5027-cbe8-40ed-ee12d1fe6495.pdf (accessed September 24, 2009)
24. Hunaidi, O.: Economic comparison of periodic acoustic surveys and DMA-based leakage management strategies. In: *Proceedings of Leakage 2005 Conference*, Halifax, N.S., Canada, pp. 322–336 (2005)
25. Maturana, F.P., et al.: Real time collaborative intelligent solutions. *SMC* (2), 1895–1902 (2004)
26. Wooldridge, M.: *An introduction to MultiAgent Systems*. John Wiley & Sons, Chichester (2002)
27. Weiss, G. (ed.): *Multiagent systems: a modern approach to distributed artificial intelligence*, p. 619. MIT Press, Cambridge (1999)
28. Stone, P., Veloso, M.: Multiagent Systems: A survey from a Machine Learning perspective. *Autonomous Robots* 8(3), 345–383 (2000)
29. Lee, R.S.T.: *Fuzzy-Neuro Approach to Agent Applications: From the AI Perspective to Modern Ontology*. Springer, Heidelberg (2006)
30. NetLogo homepage (2007), <http://ccl.northwestern.edu/netlogo/>
31. Rossman, L.A.: *EPANET 2 User's Manual*. Cincinnati (IN), USA, Environmental Protection Agency (2000)

Rateless Codes for Reliable Data Transmission over HomePlug AV Based In-Home Networks

J.P. Muñoz-Gea, P.J. Piñero-Escuer, J. Malgosa-Sanahuja, P. Manzanares-Lopez,
and J.C. Sanchez-Aarnoutse

Department of Information Technologies and Communications
Polytechnic University of Cartagena

Campus Muralla del Mar, 30202, Cartagena, Spain

{juanp.gea, pedrop.escuer, josem.malgosa, pilar.manzanares,
juanc.sanchez}@upct.es

Abstract. With the appearance of peer-to-peer networks and the rapid progress being made in the technologies used to deploy them, in-home networks are likely to play a highly important role in what is being called the Future Internet. There are different candidate technologies to widespread an in-home network. Among them, broadband communications over power line networks have attracted much interest in academy and industry recently. However, there are several aspects of the PLC medium that make it difficult to share resources fairly, such as time-varying behaviour or the broadcast nature of the channel. For these reasons TCP protocol may not be the adequate mechanism for reliable data transmission over PLC networks. In this work, the main characteristics of the PLC channel and the feasibility of using rateless Codes for reliable data transmission in these kind of networks are evaluated.

Keywords: Fountain codes, Online codes, power-line communication, HomePlug AV, home networking.

1 Introduction

Nowadays, home appliances are becoming information appliances and they can be networked to exchange their information. Therefore, it is necessary a home network able to provide support for video and data transmission from a variety of sources in the home. Candidate networking technologies to provide convenient and widespread residential networking services may be categorized as wireless, wired and no-new-wires networks. For the no-new-wires networks category, broadband communication over low voltage (220v) power lines or PLC (Power Line Communications) has attracted much interest in the academic and industrial context recently. There are different PLC technology standards, but the most popular is HomePlug AV (HomePlug Audio and Video, or simply HPAV). This standard, which was presented by Homeplug Powerline Alliance [1] in 2005, employs advanced physical and medium access control (MAC) technologies that provide a 200 Mbps power-line network. The physical layer utilizes this 200 Mbps rate to provide a 150 Mbps information rate. Nowadays, there are about 70 certified HomePlug products [4]. They range from a simple HomePlug Ethernet adapter

(which connects the Ethernet devices to the HomePlug network) to 1 Gbps high performance low-voltage PLC modem [3]. On the other hand, nowadays some companies are adding HomePlug circuits directly into multimedia home entertainment equipment. By this way, it will not be necessary any additional equipment (like Ethernet adapters) to connect the electrical appliances to the PLC network.

However, it is necessary to take into account that there are several aspects of the PLC medium that make it difficult to share resources fairly. For example, all the electronic or electrical equipments connected to the power lines are considered as noise resources on the power grid [6]. As a consequence, the HPAV modems must adapt their data rate in order to avoid packet losses. Therefore, our first goal in this work is to characterize the network behavior in presence of electrical equipment of a typical home when using a commercial HPAV modem.

Another important aspect in PLC networks is that, although the HPAV MAC layer provides a connection-oriented service based on TDMA to support QoS requirements, almost all commercial HPAV modems only support traditional connectionless, prioritized contention service based on CSMA/CA to transmit best-effort applications and applications that rely on prioritized QoS. In other words, the PLC channel is broadcast (i.e. shared) and half-duplex by nature. This fact made us think that the use of unidirectional rateless codes to transmit the information may be advantageous as opposed to traditional solutions based on bidirectional (full-duplex) TCP protocol. Moreover, TCP protocol is designed to be used over constant rate links as it chooses its parameters according to the RTT (Round trip time) value, however, as said before, the rate in PLC links can vary according to the power grid noise. Rateless codes are also known as Fountain codes because the sender is theoretically always sending coded packets. Some of them are lost in the noisy channel but when the destination node has received enough of them, the information can be completely restored. They are mainly used in multimedia applications, such as IPTV or radio broadcasting, because in these cases there is usually no feedback channel, and it is impossible to take advantage of any type of error or flow control. Online Codes are a free-software Fountain codes alternative, which achieve linear cost for encoding and decoding operations. In this work, the feasibility to use Online Codes for reliable data transmission in an in-home PLC network is evaluated.

The remainder of the paper is organized as follows. Section 2 presents the HomePlug AV specification. Section 3 introduces the encoding and decoding procedure used by Online Codes. Section 4 evaluates the use of Online Codes for data transmission in a real scenario. Finally, Section 5 concludes the paper.

2 HomePlug AV

As it was previously introduced, there are several aspects of the PLC medium that make it difficult to share resources fairly. In order to solve these problems, advanced coding and modulation mechanisms are used. The Physical Layer (PHY) operates in the frequency range of 2 - 28 MHz and provides a 200 Mbps channel rate and a 150 Mbps information rate. It uses OFDM and a powerful Turbo Convolutional Code (TCC). OFDM is a spectrum efficient modulation technique, which uses simultaneous transmission of a large

number of narrow band carriers. These carriers divide a large frequency channel into a number of subchannels. Subchannels can differ greatly in their quality, defined by their signal to noise ratio- SNR-. Adaptive coding modulation for each subchannel solves this problem by giving each subchannels an appropriate capacity, and by switching off those with a poor channel condition.

HomePlug AV provides two kinds of communication services: a connection-oriented contention free service, based on periodic Time Division Multiple Access (TDMA) allocations of adequate duration, to support the QoS requirements of demanding applications; and a connectionless, prioritized contention service, based on Collision Sense Multiple Access/Collision Avoidance (CSMA/CA), to support both best-effort applications and applications that rely on prioritized QoS. To efficiently provide both kinds of communication service, HomePlug AV implements a flexible, centrally-managed architecture. The central manager is called a Central Coordinator (CCo). The CCo establishes a beacon period and a schedule which accommodates both the cotention free allocations and the time allotted for contention-based traffic. The Beacon Period is divided into 3 regions: beacon region, CSMA region and TDMA region.

3 Fountain Codes

3.1 Description

The main idea behind Fountain codes is that the transmitter is represented like a fountain of water that is able to produce an infinite number of water drops. The receiver represents a bucket that needs to collect a number of these water drops to obtain the information. The main advantage of these codes is that the receiver can obtain the information irregardless of which drops it has collected. Therefore, Fountain codes should have the following properties:

- A transmitter can generate a potentially infinite amount of encoded packets from the original data.
- A receiver can decode a message that would require K packets from any set of K' encoded packets, for K' slightly larger than K .

The most important implementations of Fountain codes are LT codes [7], Raptor codes [10] and Online Codes [8]. LT codes were the first practical realization of a fountain code. The only drawback of these codes is that their encoding and decoding costs scale as $K \log_e K$, where K is the file size. Raptor codes are an evolution of LT that achieve linear cost for encoding and decoding. Finally, Online Codes are a free-software alternative to Raptor codes that also achieve linear cost for both operations.

There are lots of application of Fountain codes in digital communications. They are mainly used in multimedia applications, such as IPTV or radio broadcasting because in these cases there is no feedback channel. Therefore, it is impossible to take advantage of any type of error or flow control. Another kind of service that could also use these codes is multicast transmission. When a transmitter sends a file to different receivers, each of these receivers could experience independent losses, delays, jitter, etc. Without the Fountain codes facility, the number of control packets needed to maintain the multicast connection could be very high.

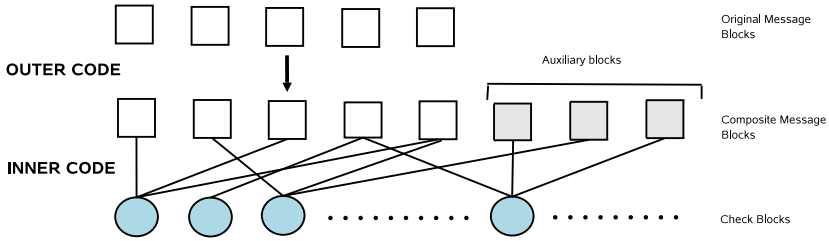


Fig. 1. Structure of Online Codes

3.2 Online Codes

Online Codes are characterized by two parameters ϵ and q . The first parameter is related to the number of coded blocks (also called *check blocks*) that the receiver needs in order to decode the original message, and the second one has an effect on the probability of succesful decoding. In particular, the receiver can recover the original message from any $(1 + 3\epsilon)K$ check blocks with a success probability determined by $1 - (\epsilon/2)^{q+1}$.

The structure of Online Codes is depicted in Fig. 1. The encoding process is divided into two layers, the inner code and the outer code. The inner code is in charge of generating the *check blocks*. Every check block is computed as the XOR operation of d blocks uniformly chosen from the message (d represents the degree of the check block). The probability that $d = i$ is given by the probability distribution ρ_i :

$$\rho_1 = 1 - \frac{(1 + 1/F)}{(1 + \epsilon)} \tag{1}$$

$$\rho_i = \frac{(1 + \rho_1)F}{(F - 1)i(i - 1)} \quad i = 2, 3, \dots, F \tag{2}$$

where F is given by

$$F = \frac{\ln(\epsilon^2/4)}{\ln(1 - \epsilon/2)} \tag{3}$$

However, due to the random selection of the message blocks, some of them may not be selected in the inner coding process. One solution to this problem is to add a preliminar coding process (called outer coding) which generates $0.55q\epsilon K$ auxiliary blocks from the original message. The message blocks that do not participate in the inner process will be able to be decoded thanks to this redundancy. In fact, the input blocks of the inner coding are the message blocks plus auxiliary blocks. All this set is called composite message.

From all the available procedures to generate redundancy (Reed-Solomon, Cyclic Redundancy Check, Parity Bits, etc.), Online Codes uses one of the simplest: for each block of the original message, q auxiliary blocks are chosen. Each auxiliary block is computed as the XOR operation of the original message blocks assigned to it.

The original message can be decoded from the check blocks, with the success probability showed before. The decoding process is also divided into two steps. In the first

step a $1 - \epsilon/2$ fraction of composite message blocks should be recovered. The knowledge of this fraction of blocks is enough to decode the original message: the composite message has such a property thanks to the redundancy added by the outer code. The second step consists of recovering the original message from the composite message blocks recovered in the first step.

In order to successfully recover the needed fraction of composite message blocks, it is necessary to get to know the degree of each check block and the composite message blocks from which it is made up (also called *adjacent blocks*). A way to send this information to the receiver must be implemented on the transmitter side. However, if the receiver uses the same random number generation algorithm as the transmitter, it will only be necessary to send the seed to reach this objective. Next, the decoding process can start. It has the following steps:

1. Find a check block with only one adjacent block ($d = 1$) and recover this composite message block.
2. Remove this recovered block from other check blocks that also have this recovered block as adjacent (by simple subtracting it; that is, computing the XOR again). After this, some check blocks can become degree-one blocks.
3. Continue with this process until a $1 - \epsilon/2$ fraction of composite message blocks is recovered.

The process can fail if in some of these steps there are no degree-one blocks.

When all the needed composite message blocks are recovered, the same process can be used to obtain the original message blocks. In this case the success probability is close to one because only the auxiliary blocks have a degree higher than one.

4 Evaluation

In this section we want to evaluate two characteristics of HomePlug AV specification: First, the variable capacity model of the physical layer; and second, the contention-based service of the MAC layer. All the evaluations presented in this section have been made using a laboratory test-bed. The lab has three phases of 220 volts, and all the PLC adapters and PC computers used in the evaluation are connected to the main phase. We have used the PLE200 HomePlug AV Ethernet adapters of Linksys [2]. This adapter connects the Ethernet device of a computer to the 220 V power line.

4.1 Evaluation of the Variable Capacity of the Physical Layer

Our first objective is to evaluate the adaptation of the data transmission rate according to the noise level. For this aim, we use two computers connected to the HomePlug AV network using the corresponding Ethernet adapters. Their power suppliers are connected to another electrical phase of the lab, different from the one used to implement the HomePlug AV network. Knowing that in Europe the common point of two distinct phases is located in the low voltage transformer, the distance between two elements connected to different phases is around 300 meters. This distance is large enough to ensure the absence of interference among devices connected to different phases.

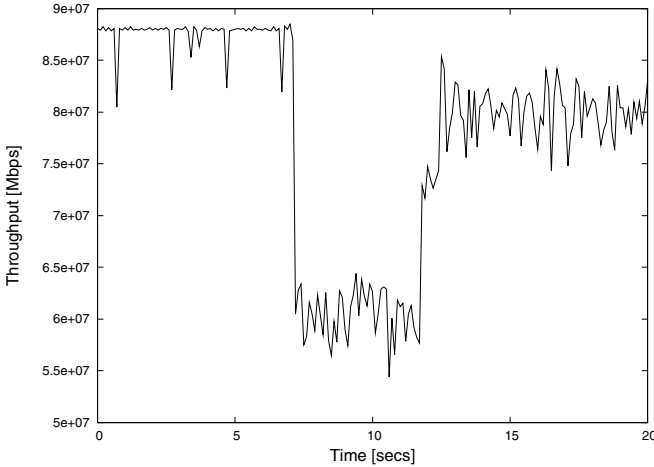


Fig. 2. Evaluation of the network capacity

In order to measure the capacity of the network, we used a UDP traffic generator which transmits traffic to the maximum capacity allowed by the physical network. We achieved the results presented in Fig. 2. In particular, it can be observed that the network is able to achieve a maximum capacity around of 87 Mbps. This transmission rate corresponds with the 100 Mbps Ethernet transmission rate of the computer, minus the HPAV physical and link layer overhead. Next, after 8 seconds, we connected a mobile phone charger to the electrical phase of the HomePlug AV network. In Fig. 2 it can be observed that the connection of the mobile phone charger to the power line causes the reduction of the network capacity to 60 Mbps. After 12 seconds, we disconnected the charger from the electrical line and we detected that the transmission rate slowly increased up to 87 Mbps again. The reason for this phenomenon is that the powerful coding and modulation technique used in HomePlug AV is able to adapt the transmission speed according to the noise level generated by the electrical devices in order to avoid the losses of packets.

The previous experiment has been repeated with other electronic devices. Table 1 represents the reduction of the capacity of the network for every device. Among the most usual devices the mobile charger and energy efficient light bulb are the noisiest. On the other hand, a simple extension lead (without any connected devices) reduces the capacity to 72 Mbps.

These sudden data rate variations cause a poor behaviour of the TCP protocol, as it defines most of its parameters according to the RTT value. This is one of the facts that make us think that the use of rateless codes for reliable data transmission over HPAV networks may be advantageous as opposed to TCP.

4.2 Evaluation of the Contention-Based Service of the MAC Layer

As it was previously introduced, the HomePlug AV MAC layer offers a contention service based on CSMA/CA. In packet communication networks, contention is a media

Table 1. Channel capacity with different electrical devices connected to the power line. 95% confidence intervals.

Device	Channel capacity [Mbps]
Without noise	86.921 ± 0.131
Mobile phone charger	60.600 ± 0.458
Laptop	84.045 ± 0.253
Wishk	82.236 ± 0.824
Heater	83.884 ± 0.142
Multimedia hard disk	86.379 ± 0.229
PC screen	76.303 ± 0.141
Electric heater	59.061 ± 0.780
Reading lamp	79.519 ± 0.118
Extension lead	72.106 ± 0.125
Energy efficient light bulb	57.058 ± 0.726

Table 2. Proportion of lost packets in UDP connections

File size [MB]	Losses [%]
1	0
2	16.66
3	36.5
4	38.17
5	30.84
6	40.77
8	34.73
10	41.45
15	37.25
20	37.73

access method that is used to share a broadcast medium. In this method all the hosts connected to the medium compete in order to transmit to the broadcast medium and they can only do that when the channel is “idle”. In order to evaluate this characteristic of HomePlug networks, in our next experiment we establish three simultaneous (compete against one another) file transmissions using UDP connections, and we evaluate the performance of one of them. The size of the transmitted file varies from 1 up to 20 Mbytes and the distance between the PLC devices of the measured connection is about 45 meters. This is approximately the longest distance between two PLC devices in a real in-home scenario (worse case). The most remarkable result is the big proportion of lost packets, represented in Table 2. It means that the UDP receiver is not able to receive the full files. These losses are mainly due to the buffer overflow at the PLC interface because the incoming packet rate is greater than the outgoing rate at which CSMA/CA can transmit packets. Therefore, in this environment the use of UDP applications which do not implement any flow control is not recommended for reliable data transmission.

4.3 Online Codes for Reliable Data Transmission

In order to avoid the losses of packets in a contention environment, like a HomePlug networks, there are two options: first, using a protocol that implements a flow control mechanism, for example TCP (Transmission Control Protocol); second, using an application that implements some kind of forward error correction, for example, the use of Fountain codes. In the following experiments, we want to compare the performance achieved by two file transmission applications, one of them uses TCP as transport protocol, and the other one uses a kind of Fountain codes (concretely Online codes).

We used the scp program [9] as a TCP-based file transmission application. On the other hand, in order to implement the application based on Online codes, we used the Online Codes implementation available in [5]. UDP transmission capability has been added to this implementation. In contrast with some real-time multimedia applications, in a file transmission application the packet loss probability must be zero (i.e., the file must be fully received). In order to achieve this with Online Codes, the decoding failure possibility must be eliminated. This is obtained by trying to decode the original information for every received block. When the process fails, the receiver waits for the next check block and it tries to decode it again. With this method the number of *check blocks* that need to be decoded is a little bigger than $(1 + 3\epsilon)K$, but the decoding failure probability is zero.

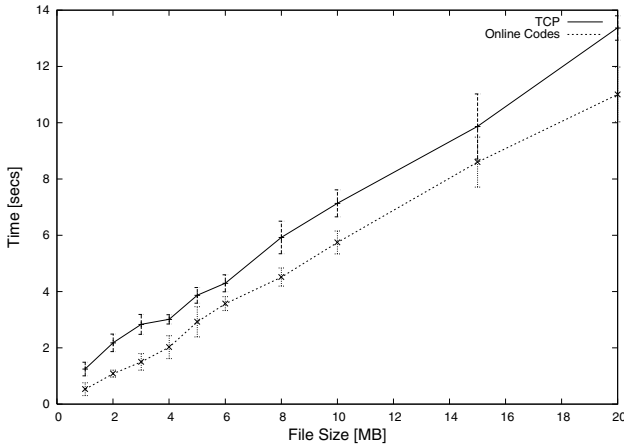


Fig. 3. Duration of TCP (scp) and Online Codes sessions in a noisy channel with two (background) data flows sharing the channel. The confidence interval has been set to 95%.

However, our application does not really decode the received blocks, it only deduces whether the received packets are enough to decode the original file, and it finishes when it has already received enough packets in order to decode it. This program implements a “light decoding” process and it gives as a result the previous amount of packets. The decoding mechanism can be implemented by another concurrent program, or it may be performed when the previous program has finished, that is, when it has received the necessary blocks to decode the original file. On the other hand, the packets can also be

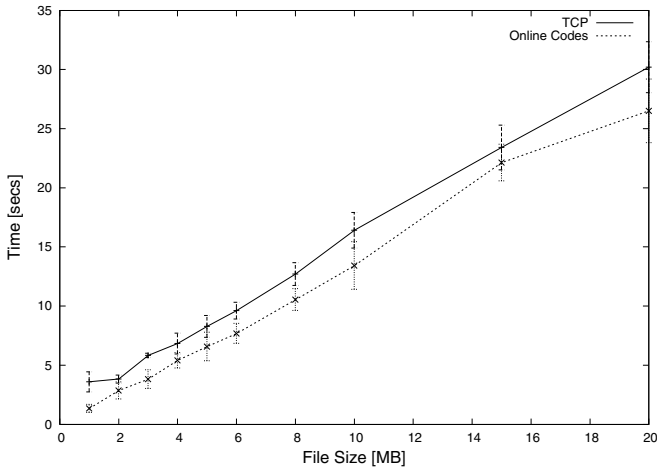


Fig. 4. Duration of TCP (scp) and Online Codes sessions in a noisy channel with four (background) data flows sharing the channel. The confidence interval has been set to 95%.

stored in a coded way. This second option could be a good choice if the coded packets are going to be retransmitted (e.g. in a P2P network) or if a OS implements a facility that is able to recognize this kind of coded files.

Both file transmission applications are evaluated in a contention scenario, with other two simultaneous file transmission connections. The size of the transmitted file varies from 1 up to 20 Mbytes. In this case, both applications are able to receive the full file without errors. Therefore, in the evaluation we are going to compare them taking into account the necessary time to receive the full file. The distance between the PLC devices of the measured connection is about 45 meters, like in the UDP scenario. Fig. 3 shows the average duration of the scp and Online Codes sessions, extracted from 5 different sessions. In addition, the figure also represents the 95% confidence interval. The first result that can be concluded from the previous figure is that the Online Codes sessions are always faster than the scp sessions. In addition, the increase of the scp sessions with respect to the Online Codes sessions is approximately constant and equal to 1 second, though for big files this increment increases up to 2 seconds. Therefore, we can draw that in a contention scenario the performance achieved by an Online Codes-based file transmission application is better than the performance achieved by a TCP-based file transmission application. Although not represented, we have also tested the FTP protocol, which also uses TCP as transport protocol. In this case, the sessions duration are always higher than with scp, and therefore the difference between the Online Codes application and FTP is also higher.

Next, both file transmission applications are evaluated again in a contention scenario, but in this case, with four simultaneous file transmission connections. Fig. 4 shows the average duration of the scp and Online Codes sessions, extracted again from 5 different sessions, and the associated 95% confidence interval. In this scenario, the Online Codes sessions are also faster than the scp sessions. However, in this case the increase of the scp sessions with respect to the Online Codes sessions is higher than in the previous

scenario, approximately equal to 2 second, and for big files this increment increases up to 4 seconds. On the other hand, if we compare these results with those presented in Fig. 3, we can observe that the increase of simultaneous file transmission connections causes an increase in the length of the sessions. This result is absolutely obvious because in this case the broadcast medium has to be shared by a bigger number of users.

As a conclusion, in multiple access networks (like in-home PLC based networks), the access control mechanism produces losses in unidirectional (like UDP) transmissions. In these cases, it is necessary to add some type of flow control (e.g. using TCP as transport protocol), or implementing some kind of forward error correction, for example, the use of Online Codes. We have compared the session lengths of a TCP file transmission application and an Online Codes based application, and we have extracted that the Online Codes sessions are always faster than the TCP sessions. Therefore, we have proved that in contention scenarios the performance achieved by Online Codes is better than the performance achieved by a TCP-based file transmission application.

5 Conclusions

Online Codes must be mainly used when the network or the application is unidirectional (broadcast-TV, satellite, IP live-TV, etc.) or in applications that cannot directly use TCP (like Application Layer Multicast). However, in a multiple access in-home network (HPAV, wireless 802.11, PhonePNA, etc.) the Online Codes are a good alternative to TCP for reliable data transmission.

Acknowledgements. This research has been supported by project grant TEC2010-21405-C02-02/TCM (CALM) and it is also developed in the framework of “Programa de Ayudas a Grupos de Excelencia de la Región de Murcia, de la Fundación Séneca, Agencia de Ciencia y Tecnología de la RM (Plan Regional de Ciencia y Tecnología 2007/2010)”. Pedro José Piñero-Escuer also thanks “Fundación Séneca” for a Séneca Program FPI pre-doctoral fellowship (Exp. 13251/FPI/09).

References

1. Afkhamie, K.H., Katar, S., Yonge, L., Newman, R.: An overview of the upcoming homeplug av standard. In: Proceedings of 2005 Internacional Symposium on Power Line Communications and its Applications (April 2005)
2. Cisco-linksys (2009), <http://www.linksysbycisco.com/>
3. Gigle semiconductor (2009), <http://www.gigle.biz/>
4. Homeplug certified products (2009), <http://www.homeplug.org/kshowcase/view>
5. Implementation of online codes (2009), <http://sourceforge.net/projects/onlinecodes/>
6. Jensen, B., Kjarsgaard, S.: Benchmarking and qos of in-house powerline equipment under noisy conditions. In: Proceedings of 2007 Internacional Symposium on Power Line Communications and its Applications (April 2007)

7. Luby, M.: Lt codes. In: Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, pp. 271–280 (2002), <http://dx.doi.org/10.1109/SFCS.2002.1181950>
8. Maymounkov, P., Mazières, D.: Rateless codes and big downloads. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 247–255. Springer, Heidelberg (2003)
9. Scp - secure copy (2009), <http://www.mksssoftware.com/docs/man1/scp.1.asp>
10. Shokrollahi, A.: Raptor codes. *IEEE/ACM Trans. Netw.* 14(SI), 2551–2567 (2006)

Replication Strategies for Business Objects in SOA

Michael Ameling¹, Bernhard Wolf¹, Thomas Springer², and Alexander Schill²

¹ SAP AG, SAP Research

Chemnitzer Str. 48, 01187 Dresden, Germany

² Technische Universität Dresden

Nöthnitzer Str. 46, 01187 Dresden, Germany

{michael.ameling,b.wolf}@sap.com,

{thomas.springer,alexander.schill}@tu-dresden.de

Abstract. Business applications, e.g., *enterprise resource planning*, can be hosted on distributed application servers within a multi-tier architecture. Business objects (BOs) are used as data containers cached at the middle-tier. The applications are replicated to achieve scalability and fast local access for the clients. Therefore, a synchronization of the BOs is mandatory to fulfill consistency requirements. Following the service-oriented architecture the synchronization of BOs through common services is time consuming and can be optimized. In this paper, replication strategies are presented that allow for an efficient synchronization of BOs based on profiles of BOs and of the hosting systems. A cost model based on an experimental evaluation allows to find the proper strategy. An initial configuration of the synchronization strategy can be adapted during runtime utilizing continuously updated profiles.

Keywords: Replication, Synchronization, Business object, Web service, SOA, Application server.

1 Introduction

Recent software solutions for mid-size companies provide functionalities of typical business applications e.g., Customer Relationship Management (CRM), Project Management (PM) and Supply Chain Management (SCM). Applications are hosted typically on application servers within a multi-tier architecture which allows resource sharing and thin client support. However, the number of clients can be very high. High activity of users can lead to low performance of the applications. Furthermore, clients can be globally distributed which results in long latencies.

Replication is a solution to achieve scalability and provide fast local access. Several instances of business applications can be hosted at one application server or might even be replicated across different application servers. Since the replicated data within the applications has to be up-to-date and consistent replica control is strongly required.

Common solutions provide replication at the database level where replica control has been investigated very well over the last two decades. However, data containers in business applications are large and complex business objects (BOs). The BOs provide services to be read, created, modified and deleted. They are involved in business

Table 1. Trade-Off Full Copy vs. Delta

Case	Sender	Network	Receiver
I.	copy full BO	transfer copy	replace BO
II.	process delta	transfer delta	integrate delta

processes and link to other BOs. Especially, data belonging to one BO is usually distributed over multiple database tables. Read and write access is provided by services and always executed in the context of business operations. A replication at database level has to operate on its schema specifications only, without any assumption about the application context the data is changed in. This results in a loss of application knowledge and the separate handling of information that belongs together. Furthermore, the application servers do not necessarily access the same database type at the persistence layer. Therefore, the replication of BOs at application level is aimed where the application context can be considered and existing services can be used. The synchronization of BOs is time consuming since BOs are complex data containers to be processed and their size leads to a large data volume to be transferred. A synchronization always takes place from the changed BO at the *primary* (called *sender*) to the replicated BO at the *secondary* (called *receiver*). The replication strategy has to be carefully selected. In example the trade-off either to send the full copy or the delta of a changed BO has to be found. The two cases are listed in Table 1. In Case I, copying the BO at the sender and a replacement at receiver side is not very time consuming. On the other hand, a message including a full copy results in long transfer time. In Case II, the sending of a delta means additional effort at sender and receiver for processing the delta of the BO and integrating the changes. However, this way the message size can be reduced significantly, if just a subset of the BO data has been changed. Therefore, transfer time can be saved compared to case I.

Our solution focuses on an adaptive synchronization for BOs at the middle-tier to support an efficient update process for replicated BOs. We consider a primary copy approach where write access is only granted to one master BO. We introduce a *profiling of BOs* and a *profiling of the system environment* to achieve an efficient synchronization process exploiting knowledge about the BO structure and application level context of change. In order to achieve efficiency we provide a *cost model* based on a *BO model* used for profiling of BOs and a *system model* used for profiling of the system environment. In Fig. 1 the conceptual steps of our approach are depicted. The first step (1) is the *profiling of BOs* based on the BO model. BO instances are profiled individually. The second step (2) is the *determination of system parameters*. The profile of each system is stored persistently as well. The third step (3) is the *execution of the cost model* where the BO profiles and system profiles are used to determine the costs for the synchronization processes based on our cost model. In step four (4) the result of step three can be used to choose and configure the replication strategy to achieve an efficient replication process. Since BOs are changing, BO and system profiles have to be updated and replication parameters have to be adjusted accordingly. Step (5) performs an *adaptation of replication parameters* during runtime.

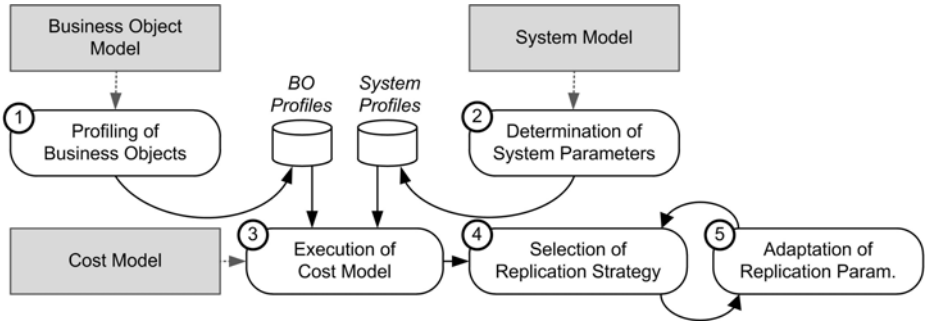


Fig. 1. System Support for Efficient Synchronization of Business Objects

The rest of the paper is organized as follows: In Section 2 the synchronization of BOs in SOA is described in detail. It is followed by related work in Section 3. Afterwards, we follow the conceptual steps in separate sections as depicted in Fig. 1: profiling of BOs in Section 4, determination of system parameters in Section 5, execution of cost model in Section 6 and the selection of the replication strategy in Section 7. In Section 8 a short conclusion and outlook is given.

2 Synchronization of Business Objects in SOA

According to a multi-tier architecture the application logic is executed at the middle-tier. At this tier, BOs are instantiated and cached for local access and efficient processing. This set-up allows implementing agents watching BOs for changes. A change pointer which references changed BOs is advisable since agents slow down the response time for clients. However, synchronization agents at sender and receiver include the replica control observing the BOs for changes, assembling synchronization messages and integrating the changes. Following a service-oriented architecture (SOA) the business logic of the applications is exposed through well defined interfaces. The same applies for replication processes such as the synchronization of BOs. At application level Web Services [17] are used to send synchronization messages via the network.

In widely distributed systems the lazy primary copy approach is often used [1]. Primary copy performs well for read intensive applications and simplifies concurrency control [6]. Modifications can be performed on BOs at the primary (master) only. Secondaries have to pass changes to the primary. The goal is to keep all BOs consistent. We assume that all BOs at secondaries have the same state. Therefore, the same synchronization message is sent to all secondaries (receivers).

Summarized, the synchronization of BOs in SOA can be distinguished in processes taking place at the sender, the transport of the synchronization message and the processes taking place at the receiver. A confirmation message is not mandatory within an optimistic approach and therefore not considered. In Fig. 2 a time based division in sender (T_S), network (T_N) and receiver (T_R) is depicted. T_S concludes the time for all processes at the sender. It starts once the transaction of a client is committed and the BO B was changed to BO B^* . It ends when the synchronization message is assembled

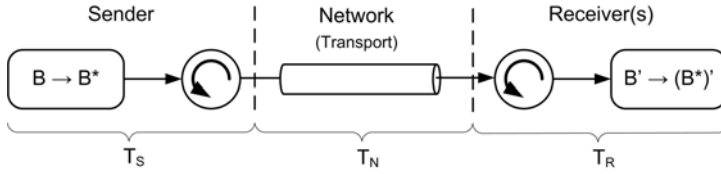


Fig. 2. Synchronization Process

and passed to the infrastructure. T_N concludes the time for the transport of the synchronization message. Several receivers can be addressed in parallel. Therefore, the longest transport time is relevant to reach consistency of the complete system. T_R concludes the time for processing the message and updating the BO B' to $(B^*)'$ at one receiver. The synchronization process is finished when all replicas have the same state as the primary.

3 Related Work

The synchronization of BOs requires an understanding of replica control at application level. We took advantage of the replication strategies well known for databases ([11], [12], [15]). Approaches implementing a replication at the middle-tier are Middel-R [9], Ganymed [14] and CORBA ([10], [7]) but mainly focus on one single algorithm or the replicated application servers share one single database. The introduced primary copy approach is used in research solutions [5], [4] as well as industrial solutions such as JBoss and WebSphere since it performs well for read intensive applications. Most approaches are primarily designed to achieve fault-tolerance ([18]). The only approaches mainly focusing on the replication of BOs are [16] and [13] but use specific algorithms. In [2] a cost model for an efficient BO replication was introduced which is included in the proposed solution. Furthermore, a framework for simulating the configuration of different replication strategies was introduced in [2].

4 Profiling Business Objects

The schema of a BO defines which elements belong to a BO, where they are placed and which elements are mandatory or optional. The choice of optional elements and cardinalities do not allow determining BO structure. In the following a BO model is introduced that allows a profiling of the structure and further parameters of BO instances.

4.1 Business Object Model

The BO model defines the parameters used for profiling BOs. It covers the parameters that influence the synchronization process. In combination with the cost model a recommendation for the configuration of replication strategies can be provided. In [2] we introduced a cost model for an efficient BO replication based on the structure of BOs. Therefore, we defined a *structure model* for BOs. The following BO model includes the

structure model as one parameter set. Further parameters are completeness, access ratio and occurrences of BOs. An introduction of all BO model parameters follows.

The *structure* of BOs allows to determine processing times for BOs at sender and receiver side. A determination on processing times based on the size of BOs only is not possible since the structure has significant influence on the overall synchronization time [2]. The structure model includes the number of elements, their size, and their position within the BO. Therefore, T_S and T_R based on the data of synchronization messages can be provided.

The *completeness* expresses the use of mandatory elements according to the schema. Therefore, empty values and non used elements can be identified. Since optional elements and cardinalities increase the use of elements an additional value for completeness including optional elements and cardinalities has to be profiled as well.

The *access* to BOs by client applications plays a significant role for the synchronization process. Read access affects priority of synchronization of BOs. A small *read ratio* might be prioritized as low. Synchronization of often read BOs can be defined with higher priority. On the other hand, the *write ratio* has influence on the amount of synchronization messages. Each write implicates a change which results in a need for synchronization.

The value *occurrences* indicates the number of BO instances existing within a system. It is the only value not related to one BO instance but to BO classes. It has influence on e.g., the number of messages to expect.

Finally, all the introduced parameters compose the BO model which is the foundation for BO profiling. Each profile of a BO holds all parameters of the BO model. An example profile is given in Table 2.

4.2 Profiling of Business Object Instances

BO instances are profiled to get the structure model parameters. The other parameters of the BO model are only determinable from BO instances as well.

In Table 2 an example of a *Sales Order* profile is given. A selection of structure model parameters [2] are listed: N - number of nodes, L - number of levels, K - number of attributes, A - maximum number of positions, W - size of all attributes in bytes, V - total size of leaf nodes, F - number of links, and a selection of N_l - number of nodes at level l . Furthermore, values for the size of the complete BO, the completeness (incl. optional elements and cardinalities in parentheses), read ratio, write ratio and occurrences are listed.

The profiling of the BO's structure can be done for all BOs right after the instantiation of the BO. During runtime profiles have to be updated when BOs were changed. The time for updating profiles can be ignored for write operations. The Algorithm 1 describes how to create a structure profile for a node. Algorithm can be used iteratively to get a complete structure profile of a node. The parameters for number of nodes (N), levels (L), attributes (K) and links (F) are collected. The size of attributes (W) and values (V) are determined as well. Furthermore, the parameters for certain nodes, positions and levels have to be collected. Therefore, the indexes l (level), n (node) and λ (position in a node) are used.

Table 2. Example Sales Order Profile

STRUCTURE							
N	L	K	λ	W	V	F	...
865	27	1,345	27	899,455	13	12	...
N_1	N_2	N_3	N_4	N_5	N_6	N_7	...
3	12	43	23	44	33	12	23
SIZE		380.233 Byte					
COMPLETENESS			63% (391%)				
ACCESS							
read ratio		3.78 h^{-1}		write ratio		1.18 h^{-1}	
OCCURRENCES			589,333				

Algorithm 1. *getProfile(nodes, level)* determination of profile parameters

Require: node n , level l

- 1: $N++$; N_l++ ;
 - 2: **if** node contains attributes **then**
 - 3: $\lambda = 0$;
 - 4: **for all** attribute k **do**
 - 5: $K++$; K_l++ ; K_n++ ; $K_\lambda++$; $\lambda++$;
 - 6: $W = W + W(k)$; $W_l = W_l + W(k)$;
 - 7: $W_n = W_n + W(k)$; $W_{n,\lambda} = W_{n,\lambda} + W(k)$;
 - 8: **end for**
 - 9: **end if**
 - 10: **if** node contains sub nodes **then**
 - 11: **if** $L < l + 1$ **then**
 - 12: $L++$;
 - 13: **end if**
 - 14: **for all** sub node u **do**
 - 15: $getProfile(u, l + 1)$;
 - 16: **end for**
 - 17: **else if** node has value **then**
 - 18: $V = V + V(v)$; $V_l = V_l + V(v)$; $V_n = V_n + V(v)$;
 - 19: **if** value is a link **then**
 - 20: $F++$; F_l++ ;
 - 21: **end if**
 - 22: **end if**
-

The update of the structure profile works similarly. Adding and deleting of elements increases and decreases parameter values. Modification of elements only affects the parameters for size of elements. The read ratio is logged during system operation. The write ratio can be determined from timestamps and versioning of BOs. We assume that the schema of a BO is available. Therefore, the completeness can be determined with a comparison of the structure profile and the BO schema.

4.3 Business Object Analysis Tool

The profiling of BOs can be done within the application or as a loosely coupled service. We developed a Business Object Analysis Tool (BOAT) that provides profiling as a Web Services. BOs can be sent as XML documents in the request. The response is a BO profile.

BOAT allows to group profiles. Therefore, it is possible to make general assumptions about BOs. In example profiles can be grouped by a BO type or even for a specific domain. A user interface providing chart views is implemented. The schemas of BOs are stored in a repository to allow a comparison and to determine the completeness.

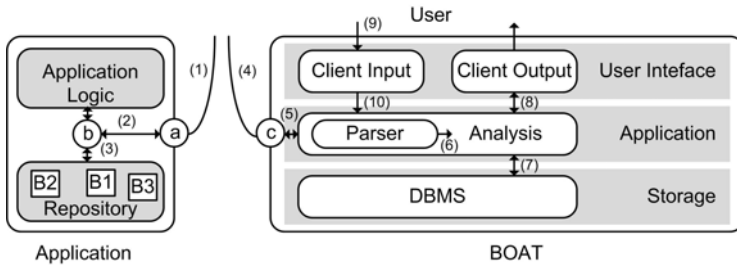


Fig. 3. Business Object Analysis Tool

In Fig. 3 the scenario for profiling BOs from an application including the architecture of BOAT is depicted. The applications allow requesting BOs through services (a) (1). The services (b) provided by the BOs (*B1*, *B2*, *B3*) themselves are used to get full copies (2) and finally return the BO instances. The BOs can be stored locally at any application. BOAT provides the Web Service (c) to profile BOs (4). Once the Web Service is called the BO document is processed (5) and an analysis is done (6). The result in form of a profile is stored persistently (7). The response of the Web Service (c) includes the profile. BOAT provides a user interfaces which allows e.g., a statistical analysis of profiles as mentioned above.

5 Determination of System Parameters

5.1 System Model

The system model provides the parameters describing the costs for a single process for synchronization. In Fig. 2 we already divided a BOs synchronization into processes at sender, network and receivers. For simplification we assume that each receiver behaves equally.

At the sender the parameters for identifying changes, parse the BO and message assembling are crucial. The first parameter describes the time an agent needs to compare a new version of a BO with the previous version of that BO until all changes are identified. The parsing of the BO can be determined by single parameters for processing each element of a BO. The parameters for the processing time for a node *a*, of an attribute *b*,

Table 3. Example System Profile

SENDER				
a_l in ns	b_λ in ns	c in ms	d in ms	e in ns
43.53 l	47.596 λ	106.24	145.87	90.684
TRANSPORT				
B_W in MBit/s	L_W in ns	R_F		
5.443	127	12		
RECEIVER				
a_l in ns	b_λ in ns	c in ms	d in ms	e in ns
21.765 l	23.798 λ	58.119	72.934	45.342

of a Byte of an attribute value c , of a Byte of an node value d and for resolving a link e are used. Further parameters are for identifying changes and message assembling e.g., creation of a SOAP message.

For the transport process we use the bandwidth B_W and latency L_W as system model parameters as introduced in [1]. Additionally the replication factor R_F is one parameter which describes the number of receivers to be addressed.

At the receiver the parameters for the processing time of the message has to be defined. Parameters for the parsing of the received synchronization message are the processing time for a node, attribute, etc. Since changes have to be integrated the parameter for this step has to be defined as well.

A higher granularity for the processes at sender and receiver side exists but is not focus in this paper. Additional processes during synchronization require further parameters. In example a process for security check can be added which requires an additional parameter. However, since the single process steps for a synchronization are executed sequentially parameters can be added easily.

5.2 Parameter Determination

The parameter determination to provide a system profile can be done before operational mode. A profile includes the costs for processes at sender and receiver. The transport parameters are included as well.

The determination of the parsing parameters was done in an experimental evaluation. In Table 3 an example of a system profile with a selection of parameters is given. Synchronization process steps at sender and receiver side have to be measured during execution. The bandwidth and latency for transportation of synchronization messages have to be determined by sending normalized messages.

In the sender profile the parameters for the processing time of a node in dependency of the level a_l , the processing time for an attribute in dependency of the position within a node b_λ and the other determined parameters are listed. We determined the parameters by measuring BO documents that were individually created. This way, we were able to solely modify the number of one type of element. Afterwards, we measured the

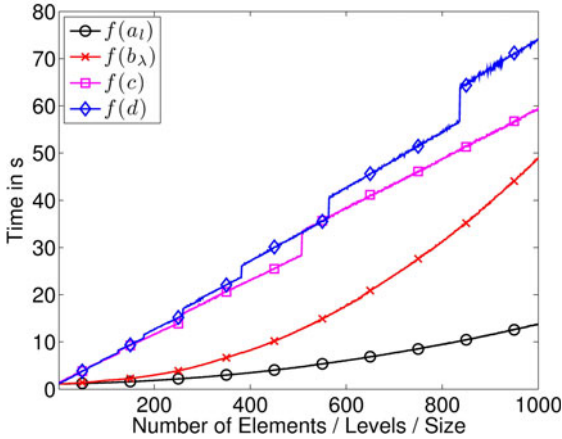


Fig. 4. System Profiling

processing time of each created BO document and were able to determine each parameter. Since different parsers are possible a comparison of different implementations parsers was done. The implementation also affects the dependency of the processing time on the structure. The listed result just present one set of the results but validate that we are able to determine the parameters with our solution.

The determination of the system profile can be done with an experimental setup which is detailed described in [2] and [3]. In Fig. 4 the measurements for increasing the number of nodes $f(a_l)$, the number of attributes $f(b_\lambda)$, the size of attribute values $f(c)$, and size of node $f(d)$ are depicted. The intersection with the ordinate is the value for processing a BO without content. The method of least squares allows determining functions out of the discrete measurements. In this way e.g., a function for the processing time of a node in dependency of the level $f(a_l)$ can be determined. The function $f(c)$ shows a linear increase for an linear increase of the attribute value. The slope exactly reflects the processing time of a Byte of an attribute value (parameter c). In Table 3 the results of the experiments for the parameter determination are listed.

6 Execution of Cost Model

In the previous sections the BO and the system model were introduced. The models are used for profiling BOs and the systems used in the replicated environment. In [2] the part of the cost model for determining the parsing time of BOs was introduced. It uses the structure parameters of the BO profile. The following equation enables to calculate the processing at the sender side:

$$T_S = T_{offset} + \sum_{l=0}^L (N_l \times a_l) + \sum_{\lambda=1}^{K_n} (K_\lambda \times b_\lambda) + W \times c + V \times d + F \times e$$

The defined cost model presents a sum of the time needed for all process steps for one synchronization of a changed BO. The lazy primary copy approach for replication is used. Therefore, we consider the synchronization from a master to the replicas.

However, our cost model does also apply for the use of other replication strategies. In example, a termination strategy does require an additional confirmation message resulting in additional message assembling at receiver side, an additional transport of that message and a processing at the sender. A switch to the update every approach can be covered [6]. Additional process steps can be easily inserted into the sequence of synchronization process steps. A comparison between different replication strategies is planned for future work. The current focus is the configuration of the currently used replication strategy.

The validation of the cost model was done experimentally with the use of dumps of BOs. BOAT was used to do a profiling of BOs. To avoid an implementation within a live system the replication environment introduced in [2] was used. Therefore, we were able to determine processing time of BOs and to simulate a synchronization of changed BOs. The determination of read and write ratio usually requires a comprehensive observation of representative applications.

7 Selection of Replication Strategy

The cost model including profiling is used to support an efficient configuration of the synchronization process. The following two examples for the selection of the replication strategy are given: the sending of a *full copy* of a BO verses the sending of the *delta* (amount of changes between two different BO versions) and the *bulking* of BO changes. Both strategies can be combined.

7.1 Full Copy vs. Delta

To reach more efficiency the time efforts for *case I* and *case II* have to be compared (Table 1) to choose the most beneficial. Therefore, the processing times T_P and the transfer times T_N for both cases have to be determined. The processing time T_P summarizes the process steps of the synchronization at the sender (T_S) and at receiver (T_R). All values for *case I* ($T_P(full)$, $T_N(full)$) and *case II* ($T_P(delta)$, $T_N(delta)$) can be determined with the introduced cost model based on profiling.

The decision either to use *case I* or *case II* depends on the trade-off between the time that can be saved at transfer (ΔT_N) and the additional effort for processing (ΔT_P). In Fig. 5 a method to describe the trade-off and to find the break-even-point to switch between the replication strategy is depicted. The method considers the relative size of the delta compared to the full size of the BO. The function $\Delta T_P(delta)$ defines the processing time that can be saved sending a *full copy* (*case I*) where $\Delta T_P(delta) = T_P(delta) - T_P(full)$. The function $\Delta T_N(delta)$ defines the transfer time that can be saved sending a *delta* message (*case II*) where $\Delta T_N(delta) = T_N(full) - T_N(delta)$. For simplification we assume that the used BO has a fixed size. Therefore, both functions depend only on the size of the delta since the size of a full copy is constant if no additional elements are added to the BO ($T_N(full) = const.$).

The function $\Delta T_N(delta)$ decreases with a larger delta size since less transfer time can be saved sending a delta message. The intersection with the ordinate is the transfer time of a full copy $\Delta T_N(0) = T_N(full)$. It equals the maximum transfer time that can

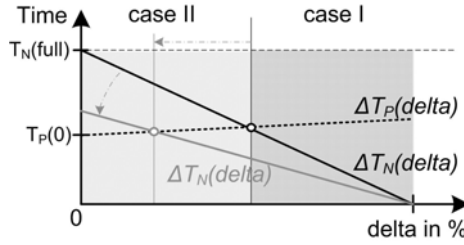


Fig. 5. Decision Model

be saved. No time can be saved when the delta equals a full copy ($\Delta T_N(100\%) = 0$). The function $\Delta T_P(\text{delta})$ slightly increases since a larger delta results in more effort for processing in *case II*. The intersection with the ordinate is the processing time for an empty delta message ($T_P(\text{delta})$). Finally, the intersection of the functions $\Delta T_N(\text{delta})$ and $\Delta T_P(\text{delta})$ equals the delta value to switch between the two cases. If the saved transfer time $\Delta T_N(\text{delta})$ sending a delta exceeds the saved additional effort $\Delta T_P(\text{delta})$ sending a full copy then *case II* is more efficient (*case II*: $\Delta T_N(\text{delta}) > \Delta T_P(\text{delta})$ - left side). *Visa versa case I* is used if $\Delta T_P(\text{delta})$ exceeds $\Delta T_N(\text{delta})$ (*case I*: $\Delta T_P(\text{delta}) \geq \Delta T_N(\text{delta})$ - right side).

The previous example was given for one BO with a fixed size. The method to find the trade-off considers the relative delta. The structure of the BO was fixed and is reflected in the cost model. Let's assume another BO with a smaller size is changed. For simplification the function for the saved processing time $\Delta T_P(\text{delta})$ does not change due to e.g., more complex structure. The smaller size of the BO results in a decrease of $T_N(\text{full})$. The function $\Delta T_N(\text{delta})$ is below the previous function (gray line in Fig. 5). Finally, the intersections of the functions $\Delta T_P(\text{delta})$ and $\Delta T_N(\text{delta})$ moves to a smaller delta resulting in an earlier switch from *case II* to *case I*. The introduced cost model and the profiling allow predicting both functions for any BO. Finally, we are able to decide either to send immediately a full copy or a delta message for each BO based on the size of the delta and the structure of the BO.

7.2 Bulking

A high write ratio of BOs causes frequent synchronization messages which can result in high network traffic. Consistency constraints allow that not synchronized BOs are still valid for e.g., a certain amount of time. Therefore, several changes can be bundled in one message. The so called *bulking* enables to reduce the traffic and the overhead that is needed for each single message. The cost model allows predicting the time needed for a synchronization process. Therefore, we are able to determine the latest possible point in time to send the synchronization message for a certain change. Currently bulking of delta messages containing independent changes is considered, i.e. no change will be overwritten. Full-copy update messages and overlapping changes are already examined but not described in this paper.

In [8] coherency predicates for consistency are introduced. These are *version distance*, *value divergence* and *temporal distance*. The predicates define if a BO replica is

still valid after a master was changed. The distance between the BO versions, the differences of the BO content or just a certain amount of time must not exceeded. The bulking of changes results in a delay for synchronization messages. Therefore, the temporal distance Δt for the replicas is crucial. It must not be exceeded to fulfill the consistency constraints.

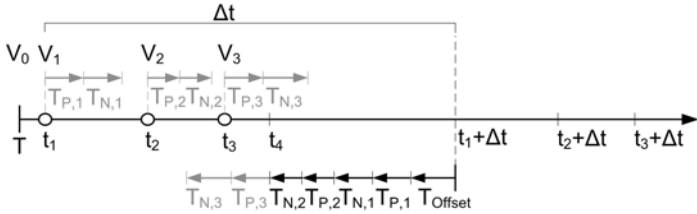


Fig. 6. Delay of Synchronization Message

For simplification we assume that Δt is constant for all BOs. The temporal distance defines the maximum time until a change of a BO has to be incorporated at all replicas. In Fig. 6 three changes of a BO at random time are depicted. The version of the BO changes from V_0 to V_1 to V_2 to V_3 . The changes are committed at the times t_1 , t_2 and t_3 . Once a change was committed the temporal distance for each change must not exceed $t_1 + \Delta t$ for V_1 , $t_2 + \Delta t$ for V_2 , and $t_3 + \Delta t$ for V_3 .

The assembling and disassembling of the message header and the transfer of the overhead is necessary for each message and takes a fixed amount of time T_{Offset} (called offset). In Fig. 6 the bulking of the changes of V_1 and V_2 is depicted. Both changes are included in one synchronization message. Due to the temporal consistency restriction the time $t_1 + \Delta t$ must not be exceeded. Therefore, we have to consider the time that is needed to process and transfer the message. Additionally to T_{Offset} the times for processing the synchronization message of the first change $T_{P,1}$ as well as the transfer time $T_{N,1}$ are necessary. The times for the second change are $T_{P,2}$ and $T_{N,2}$. An additional offset for the second change is not necessary because it is sent within the same synchronization message. All values can be determined with the help of the cost model. The synchronization message can be sent at the latest at $(t_1 + \Delta t) - (T_{Offset} + T_{P,1} + T_{N,1} + T_{P,2} + T_{N,2})$ which equals t_4 .

In the example, the third change at t_3 cannot be included in the same synchronization message. The sum of all processing times $T_{P,1}, T_{P,2}, T_{P,3}$, all transfer times $T_{N,1}, T_{N,2}, T_{N,3}$ and the offset exceeds the time left between t_3 and $t_1 + \Delta t$. Bulking all three changes into one synchronization message violates the temporal distance Δt for the first change V_1 .

8 Conclusions

This paper discusses an approach for adaptive synchronization of business objects replicated at the middle-tier. We introduced a profiling for BOs and system parameters. Profiling allows determining the processing and transfer costs for the synchronization. The

sending of full copies of BOs or delta synchronization messages as well as temporal consistency constraints are considered. A cost model based on an experimental evaluation allows configuring the used replication strategy to achieve an efficient synchronization. A validation was done by profiling real BO instances and the implementation of a simulation environment. The introduced approach of adaptive synchronization is applicable for an initial configuration of the replication strategy for BOs and an adaption during runtime.

References

1. Ameling, M., Roy, M., Kemme, B.: Replication in service oriented architectures. In: IC-SOFT, pp. 103–110 (2008)
2. Ameling, M., Wolf, B., Armendariz-Inigo, J.E., Schill, A.: A cost model for efficient business object replication. In: AINA Workshops 2009, pp. 304–309 (2009)
3. Ameling, M., Wolf, B., Springer, T., Schill, A.: Experimental evaluation of processing time for synchronization of xml-based business objects. In: Bhowmick, S.S., Küng, J., Wagner, R. (eds.) DEXA 2009. LNCS, vol. 5690, pp. 255–265. Springer, Heidelberg (2009)
4. Barga, R., Lomet, D., Weikum, G.: Recovery guarantees for general multi-tier applications. In: ICDE (2002)
5. Felber, P., Narasimhan, P.: Reconciling replication and transactions for the end-to-end reliability of CORBA applications. In: (DOA) (2002)
6. Gray, J., Helland, P., O’Neil, P., Shasha, D.: The dangers of replication and a solution. In: SIGMOD, pp. 173–182 (1996)
7. Killijian, M.-O., Fabre, J.C.: Implementing a reflective fault-tolerant CORBA system. In: SRDS (2000)
8. Lenz, R.: Adaptive distributed data management with weak consistent replicated data. In: SAC 1996, pp. 178–185. ACM, New York (1996)
9. Marta Pati, N.-M., Jiménez-Peris, R., Kemme, B., Alonso, G.: Middle-r: Consistent database replication at the middleware level. ACM Trans. Comput. Syst. 23, 375–423 (2005)
10. Othman, O., O’Ryan, C., Schmidt, D.C.: Strategies for CORBA middleware-based load balancing. In: IEEE Distributed Systems (2001), <http://www.computer.org/dsonline>
11. Pacitti, E., Minet, P., Simon, E.: Fast algorithm for maintaining replica consistency in lazy master replicated databases. In: VLDB, pp. 126–137 (1999)
12. Pedone, F., Guerraoui, R., Schiper, A.: The database state machine approach. Distributed and Parallel Databases 14(1), 71–98 (2003)
13. Perez-Sorrosal, F., Patiño-Martínez, M., Jiménez-Peris, R., Kemme, B.: Consistent and scalable cache replication for multi-tier j2ee applications. In: Cerqueira, R., Pasquale, F. (eds.) Middleware 2007. LNCS, vol. 4834, pp. 328–347. Springer, Heidelberg (2007)
14. Plattner, C., Alonso, G.: Ganymed: Scalable replication for transactional web applications. In: Jacobsen, H.-A. (ed.) Middleware 2004. LNCS, vol. 3231, pp. 155–174. Springer, Heidelberg (2004)
15. Plattner, C., Alonso, G., Özsu, M.T.: Extending DBMSs with satellite databases. The VLDB Journal (2007)
16. Salas, J., Perez-Sorrosal, F., Marta Pati, N.-M., Jiménez-Peris, R.: Ws-replication: a framework for highly available web services. In: WWW (2006)
17. W3C. Web services (2002), <http://www.w3.org/2002/ws/>
18. Wu, H., Kemme, B.: Fault-tolerance for stateful application servers in the presence of advanced transactions patterns. In: (SRDS) (2005)

A Hybrid Approach for Database Replication: Finding the Optimal Configuration between Update Everywhere and Primary Copy Paradigms

M. Liroz-Gistau¹, J.R. Juárez-Rodríguez¹, J.E. Armendáriz-Íñigo¹
J.R. González de Mendivil¹, and F.D. Muñoz-Escóí²

¹ Dpto. de Ing. Matemática e Informática, Universidad Pública de Navarra
Campus de Arrosadía s/n, 31006 Pamplona, Spain

² Instituto Tecnológico de Informática, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain

{miguel.liroz,jr.juarez,enrique.armendariz,mendivil}@unavarra.es
fmunyo@iti.upv.es

Abstract. Database replication has been subject of two different approaches, namely primary copy and update everywhere protocols. The former only allows performing update transactions in the primary replica, while the rest are only used to execute read-only transactions. Update everywhere protocols, on the other hand, allow the system to schedule update transactions in any replica, thus increasing its capacity to deal with update intensive workloads and overcoming failures. However, synchronization costs augment and its throughput may fall below the ones obtained by primary copy approaches. Under these circumstances, we propose a new database replication paradigm, halfway between primary copy and update everywhere approaches, which improve system's performance by adapting its configuration depending on the workload submitted to the system. The core of this approach is a deterministic replication protocol which propagate changes so that broadcast transactions are never aborted. We also propose a recovery algorithm to ensure fault tolerance.

1 Introduction

Database replication is considered as a joint venture between database and distributed systems research communities. Each one pursues its own goals: performance improvement and affording site failures, respectively. These issues bring up another important question that is how different replicas are kept consistent, i.e. how these systems deal with updates that modify the database state. During a user transaction lifetime it is a must to decide in which replica and when to perform updates [8]. We focus on eager solutions and study the different alternatives that exist according to where to perform updates.

The primary copy approach allows only one replica to perform the updates [5,16]. Changes are propagated to the secondary replicas, which apply them in an ordered fashion. Data consistency is trivially maintained since there is only one server executing update transactions. Secondaries are just allowed to execute read-only transactions. This approach is suitable for workloads dominated by read-only transactions, as it tends to

be in many modern web applications [5,16]. However, the primary replica represents a bottleneck for the system when dealing with a large amount of update transactions and, furthermore, it is a single point of failure. The opposite approach, called update-everywhere [14,11], consists of allowing any replica to perform updates. In this way, system's availability is improved and failures can be tolerated. Performance may also be increased, although a synchronization mechanism is necessary to keep data consistent. This may suppose a significant overload in some configurations.

Several recent eager update-everywhere approaches [11,13,14,20] take advantage of the total-order broadcast primitive [4]. Certification-based and weak-voting protocols are the ones which obtain better results [19]. Among them, certification-based algorithms decide the outcome of a transaction by means of a deterministic certification test, mainly based on on a log of previous committed transactions [14,20,6]. On the contrary, on weak-voting protocols the delegate replica decides the outcome of the transactions and informs the rest of the replicas by sending a message in an additional round. In an ideal replication system all message exchange should be performed in one round (as in certification-based) and delivered writesets should be committed without storing a redundant log (as it is done in weak-voting).

In this paper we propose a novel approach that circumvents the problems of the primary-copy and update-everywhere approaches. Initially, a fixed number of primary replicas is chosen and, depending on the workload, new primaries may be added or removed by sending a special control message. A deterministic mechanism governs who acts as the primary at a given time. Thus, at a given time slot, only those update transactions coming from a given replica are allowed to commit: A primary replica applies the writesets in order (aborting local conflicting transactions if necessary), and when its turn arrives, local transactions waiting for commit are committed and their writesets broadcast to the rest of the replicas. This avoids the need of extra communication rounds and the access to the log in order to certify transactions. Moreover, replicas configuration can be modified dynamically both by changing the role of existing replicas (turning a primary into a secondary or vice versa) or by adding new secondaries.

Fault tolerance issues are also treated in this work, since a recovery protocol is also presented. This protocol guarantees that a (re)joining replica will recover its missed state and then continue normal processing of transactions without the need of several rounds to accelerate the process [12,17,18]. One of the most important features of the replication protocol is that received remote transactions have always to be applied without further verification. This is very attractive from the point of view of recovery since it makes normal application of remote transactions as faster as the application of missed transactions in the recovery process, avoiding the use of extra rounds to speed up recovery.

If we assume that the underlying DBMS at each replica provides Snapshot Isolation (SI) [2], the proposed protocol will provide Generalized SI [6,7] (GSI). The rest of this paper is organized as follows: Section 2 depicts the system model. The replication protocol is introduced in Section 3 and fault tolerance is discussed in Section 4. Experimental evaluation is described in Section 5. Finally, conclusions end the paper.

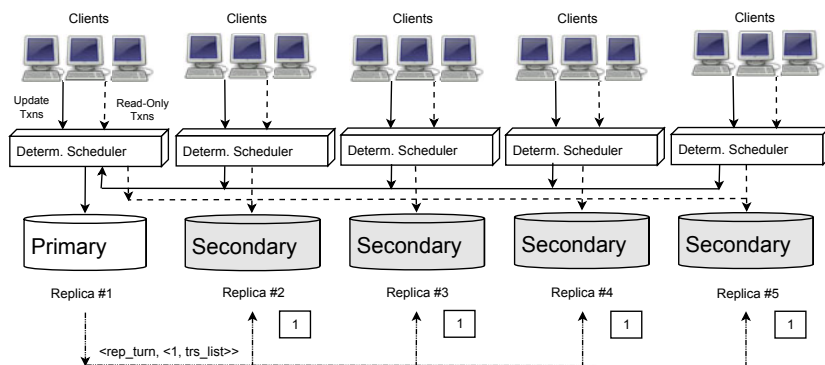


Fig. 1. Startup configuration with one primary and main components of the system

2 System Model

We assume a partially synchronous distributed system where message propagation time is unknown but bounded. The system consists of a group of sites $M = (R_0, \dots, R_{M-1})$, N primary replicas and $M - N$ secondaries, which communicate by exchanging messages. Each site holds an autonomous DBMS providing SI that stores a physical copy of the replicated database schema, i.e., we consider a full-replicated system. An instance of the replication protocol is running on each replica over the DBMS. It is encapsulated at a middleware layer that offers consistent views and a single system entry point through a standard interface, such as JDBC. Middleware layer instances of different sites communicate among them for replica control purposes.

A replica interacts with other replicas by means of a Group Communication System [4] (GCS) that provides two multicast primitives, reliable multicast and uniform multicast; and a FIFO point-to-point channel between every pair of replicas. This GCS includes also a membership service which monitors the set of participating replicas and provides them with consistent notifications in case of failures, either real or suspected. The service is provided by means of views, which represent the set of active and connected process at a particular moment. Changes on those sets are notified through view change events. Sites may only fail by crashing, i.e., Byzantine failures are excluded. They may later recover and rejoin the system, triggering a recovery procedure whose aim its to apply the missed state.

The membership service runs under the virtual synchrony model [3,4] with the sending view delivery [4] and primary-partition properties. In order to simplify the recovery procedure [12,10], we consider an extension based on the enriched virtual synchrony model [1]. Views processes are grouped in three different subviews: pr_sv , which comprises all primary replicas; sec_sv which includes all secondaries; and rec_sv , containing the rest of processes, which are recovering from a failure. Changes in the composition of views are triggered by the GCS, while changes in the compositions of subviews within a given view are requested by one of its processes.

Clients access to the system through their delegate replicas to issue transactions. The way the delegate replica is chosen depends on the transaction type. A transaction is composed by a set of read and/or write operations ended either by a commit or an abort operation. A transaction is said to be read-only if it does not contain write operations and an update one, otherwise. Read-only transactions are directly executed (and committed, without any further coordination) over primary or secondary replicas, while update ones are forwarded to the primaries where their execution is coordinated by the replication protocol.

3 Replication Protocol

3.1 Extending Primary-Copy Approach

In this paper, we extend the primary copy approach to improve its performance (mainly increasing the capacity of handling a high number of updates) and its fault tolerance. This new approach allows different replicas to be primaries alternatively (and hence to execute updates) during given periods of time by means of a deterministic protocol. As pointed out before, this protocol follows at each replica (primary or secondary) the most straightforward scheduling policy: at a given slot, only those writesets coming from a given primary replica are allowed to commit. In the primaries, other conflicting local concurrent transactions should be aborted to permit those writesets to commit [15]. Secondary replicas do not raise this problem since they are not allowed to execute update transactions (read-only transactions do not conflict under SI).

3.2 Protocol Description

In the following, we explain the operation of the deterministic protocol executed by the middleware at a primary replica R_k (Algorithm 1) Lines 9,10,29 and 30 are necessary for the recovery protocol and will be explained later in Section 4. In a nutshell, it is a replication protocol where replicas multicast their transactions in turns, so that it is ensured that when a transaction is sent it can be safely committed. Sites order themselves in a circular sequence and multicast their transactions in an ordered fashion; hence, the algorithm can be seen as a sort of round robin based protocol, where, at a given turn, only one replica acts as a primary. However, the commit of transactions is decoupled from its sending; in this way, the system is not limited by the throughput of the slower site. Prior to the sending of a transaction, it is checked that it will not conflict with any of the transactions that are pending to commit; hence, ensuring that the transaction will not be aborted.

All operations of a transaction t are submitted to the middleware of its delegate replica R_k . All except the commit request are simply forwarded to the DBMS for its execution and their responses are sent back to the client (lines 2-3). When a commit request is received, the protocol has to ensure that consistency and isolation are maintained; thus, its execution is delayed. Since, under SI, read-only transactions never cause conflicts; thus, they are straightforwardly committed (line 5). In the case of update transactions, they are stored in *trs_to_send* for further propagation (line 6). Note

Algorithm 1. Replication

```

1: upon operation request op for t from local client
2:   if op is SELECT, UPDATE, INSERT or DELETE then
3:     execute operation and return to client
4:   else if op is COMMIT then
5:     if t's writeset is empty then commit t and return to client
6:     else append t to trs_to_send
7: upon receiving message  $\langle \text{rep\_turn}, \langle \text{turn}, \text{trs\_list} \rangle \rangle$  from  $R_j$ 
8:   include  $\langle \text{turn}, \text{trs\_list}, R_j \rangle$  in reorder
9:   if turn > max_rcv_turn then update max_rcv_turn and max_rcv_turn_site
10:  if turn is the first turn after rejoining then set last_turn_to_recover to turn
11: upon  $\langle \text{last\_in\_pending} + 1, \text{trs\_list}, R_j \rangle$  is in reorder
12:  append  $\langle \text{last\_in\_pending} + 1, \text{trs\_list} \rangle$  to pending
13:  remove  $\langle \text{last\_in\_pending} + 1, \text{trs\_list}, R_j \rangle$  from reorder
14:  increment last_in_pending
15:  if  $R_j$  is my predecessor then
16:    set my_turn to true
17: upon my_turn is true
18:   remove from trs_to_send transactions which conflict with others in pending
19:   UniformMcast  $\langle \text{rep\_turn}, \langle \text{last\_in\_pending} + 1, \text{trs\_to\_send} \rangle \rangle$ 
20:   empty trs_to_send
21:   set my_turn to false
22: upon  $\langle \text{turn}, \text{trs\_list} \rangle$  is the first in pending
23:   for each transaction  $t \in \text{trs\_list}$  do
24:     remove from trs_to_send and abort transactions which conflict with t
25:     if t is local then
26:       commit t and return to client
27:     else
28:       apply t and commit
29:   store  $\langle \text{turn}, \text{trs\_list} \rangle$  in the log
30:   increment last_applied_turn
31:   remove  $\langle \text{turn}, \text{trs\_list} \rangle$  from pending

```

that for secondary replicas, only the first situation is possible; hence, there is no need to maintain *trs_to_send* variable.

At a given time, only one process is allowed to multicast the transactions that have requested commit. We refer to that process as *turn master*, and it acts as a primary for that turn. Replicas send transactions in turns according to a predefined sequence; in particular, primary replicas are ordered according to their identifiers and the list is traversed in a circular way. Since there is no guarantee in the message ordering, the processing of messages is divided into two steps. When the message is received (lines 7-10) it is first stored, together with its sender, in a special set, called *reorder*, which acts as a reordering buffer. When holes are filled (lines 11-16), the messages can be transferred to another queue, *pending*, which stores the received turns pending to be applied in the database in an ordered fashion. Moreover, when that transfer takes place, the replica checks whether it is the next one in the sequence of primaries (secondary

replicas may skip this step). If so, it becomes the turn master and it can multicast transactions in *trs_to_send* (lines 17-21). Each replica maintains both the list of primary and secondary replicas that compose the system at each moment; therefore, it can determine easily whether a given replica is its predecessor in the sequence.

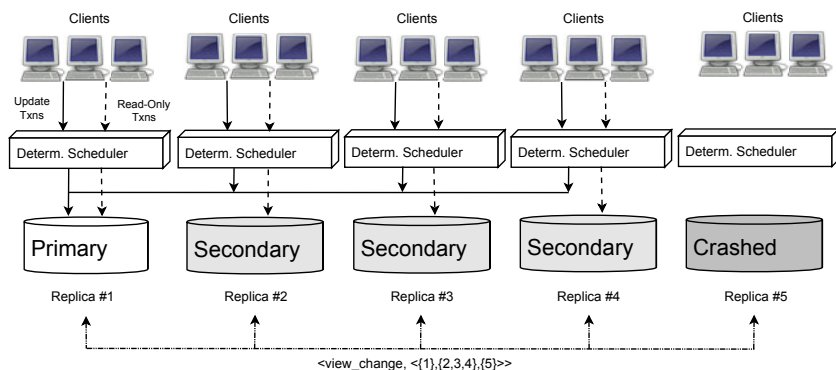
Turns stored in *pending* queue are consumed asynchronously with respect to the sending of messages (lines 22-31). For each transaction belonging to a given turn, firstly, conflicting transactions in *trs_to_send* are removed and aborted. Next, if the transaction is local, it is straightforwardly committed and the commit response is sent back to the client. If, otherwise, the transaction is remote, its changes have to be previously applied and then committed. If a DBMS with the *first updater wins* (e.g., PostgreSQL [9]) rule is used, progress must be ensured by a block detection mechanism, as the one presented in [15], which aborts all local conflicting transactions that may be blocking the remote transactions. When a DBMS with the *first committer wins* rule is used, such an extension is not necessary.

As it has been pointed out before, a specific feature of this protocol is that multicast transactions are never aborted. Let us see how this property works. When a process is allowed to multicast its transactions (line 19), it has already received all the transactions that must be committed prior to them. These transactions are the only ones that may cause the abort of the transactions being sent. The received transactions which have been already committed have aborted in their commit process all local concurrent conflicting transactions (line 24). Moreover, local transactions which had already requested their commit have been also removed from *trs_to_send* in that procedure. Then, the transactions that have survived in *trs_to_send* might only be aborted by the transactions that have been received but not committed yet. However, before local transactions in *trs_to_send* are allowed to be multicast, a sort of small certification is made and the ones that conflict with the received transactions that are pending to be committed (those in *pending*) are also removed from *trs_to_send* (line 18). That ensures that the transactions that are finally multicast are never going to be aborted.

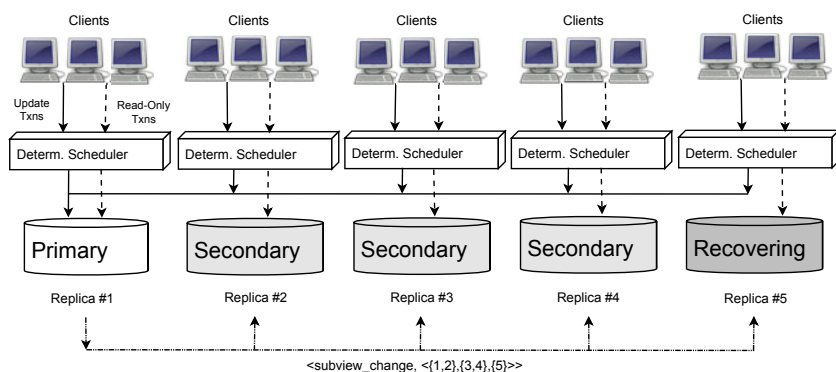
3.3 Dynamic Load-Aware Replication Protocol

Initial system configuration sets the number of primary and secondary replicas which compose the replicated system. However, this is not a fixed configuration. Our protocol may easily adapt itself dynamically to different transaction workloads by turning primaries into secondaries and vice versa. This makes it possible to handle different situations. Note that a great number of primary replicas increases the overhead of the protocol, since delay between turns is increased and there are more update transactions from other primary replicas that need to be locally applied. Therefore, it is clear that this leads to higher response times of transactions.

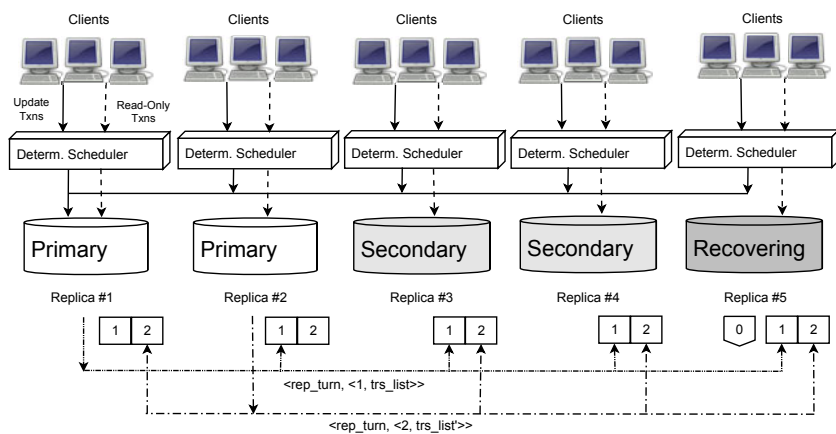
This feature improves the system capacity to handle workloads predominated by update transactions. On the other hand, increasing the number of secondary replicas does not involve a major problem, since data consistency is trivially maintained in these replicas as they are only allowed to execute read-only transactions. Thus, this improves the system capacity to handle this type of transactions, although it does not enhance the possibility of handling update ones or putting up with failures of a single primary. Therefore, the system performance is a trade-off between the number of primaries and



(a) A failed replica rejoins the system



(b) A secondary replica becomes primary



(c) The system continues processing with two primaries

Fig. 2. System reconfiguration

the number of secondaries, depending on the workload characteristics. This elasticity ensures the most appropriate configuration for each moment in terms of resources and

operational cost when compared to an under-utilized static system that over-provisions for peak update load.

In this way, our protocol is able to elastically adapt itself to the particular behavior of the workload processed in the replicated system. For this, we can think in a monitoring process at a given primary replica. Every replica periodically sends messages to this process that contains information about its load, i.e., the percentage of update transactions (empty in the case of a secondary), the number of transactions processed and other system usage parameters. The monitoring process aggregates this information across the entire system over a period of time to infer the load characteristics. We think that the simplest mechanism is to give a threshold to add, or respectively remove, primaries in the system. Let us consider a set of replicas where one is the primary and the others are secondaries, we can turn a secondary easily into a new primary in order to handle better a workload where update transactions become predominant (see Figure 2b-c). For this, it is only necessary that a primary replica request a change in subview composition, so that the *pr_sv* subview contain the new primary. When the corresponding subview change is received, the selected replica will detect that it now belongs to the set of primaries and act as one of them. In the same way, when the workload becomes dominated by read-only transactions, we can turn a primary replica into a secondary one through a similar process that updates the number of primaries and removes the corresponding entry in the working queue at each replica of the system. Finally, we would like to point out that the study and implementation of different elasticity policies (apart from the threshold one) is not the aim of this paper and this protocol simply provides the required mechanisms to reconfigure the system.

4 Fault Tolerance

In the system treated so far, replicas may fail, rejoin or new replicas may come to satisfy some performance needs. The proposed approach deals also with these issues. In this way, the protocol is extended with the actions presented in Algorithm 2, which comprises all tasks needed in case a replica fails, a replica recovers or the subviews compositions changes. Note that we differentiate regular view changes (lines 1-12) from the view changes that take place within a given view because of variations on the composition of its subviews (lines 13-14).

When a replica fails, a view change is received and it is not included in the new installed view. If the failed replica is the turn master, the next primary has to continue the sequence of turns (lines 3-4). If, moreover, there are no primaries in the new view, secondary replicas has to initiate a procedure by which some of them will promote to primaries (lines 5-6). The liveness of the algorithm is ensured provided that there is a primary view with at least one node up-to-date (either primary or secondary).

To perform the recovery procedure, it is necessary to store at each replica a log indicating which turns have been already applied in the database. It is updated whenever a replica applies and commits all transactions of a given turn (line 29 of Algorithm 1). This log is persistent and its contents are maintained even if the replica fails.

When a replica (re)joins the system, it fires a view change that initiates a recovery procedure. The recovering replica has to obtain the last applied turn from the log and

Algorithm 2. Recovery

```

1: upon receiving  $\langle \text{view\_change}, \langle \text{pr\_sv}, \text{sec\_sv}, \text{rec\_sv} \rangle \rangle$ 
2:   update primaries_subview and secondaries_subview
3:   if I am primary and max_recv_turn_site is my predecessor then
4:     set my_turn to true
5:   else if I am secondary and all primaries have left then
6:     start a procedure to elect a new set of primaries
7:   else if I am recovering and recoverer has left then
8:     ReliableMcast  $\langle \text{rec\_request}, \langle \text{last\_applied\_turn}, \text{last\_turn\_to\_recover} \rangle \rangle$ 
9:   else if I am new in view V then
10:    get last_applied_turn from the log
11:    set last_turn_to_recover to unknown
12:    ReliableMcast  $\langle \text{rec\_request}, \langle \text{last\_applied\_turn}, \text{last\_turn\_to\_recover} \rangle \rangle$ 
13: upon receiving  $\langle \text{subview\_change}, \langle \text{pr\_sv}, \text{sec\_sv}, \text{rec\_sv} \rangle \rangle$ 
14:   update primaries_subview and secondaries_subview
15: upon receiving message  $\langle \text{rec\_request}, \langle \text{begin\_turn}, \text{end\_turn} \rangle \rangle$  from  $R_j$ 
16:   if I am the selected recoverer then
17:     if end_turn is unknown then set end_turn to last_view_turn
18:     for turn  $\leftarrow$  begin_turn to end_turn do
19:       get trs_list from log for turn
20:       SendFIFO  $\langle \text{rec\_turn}, \langle \text{turn}, \text{trs\_list} \rangle \rangle$  to  $R_j$ 
21: upon receiving message  $\langle \text{rec\_turn}, \langle \text{turn}, \text{trs\_list} \rangle \rangle$ 
22:   for each transaction  $t \in \text{trs\_list}$  do
23:     apply t and commit
24:   store  $\langle \text{turn}, \text{trs\_list} \rangle$  in the log
25:   increment last_applied_turn
26:   if it is the last message of recovery then
27:     join the secondaries_subview

```

send a message to all replicas in order to obtain a recoverer. This recoverer may be selected by the recovering replica among the up-to-date nodes or it may be selected by a deterministic procedure executed at each replica. In any case, the recoverer will send back the turns in order to the recovering replica, which will apply them in the database in an ordered fashion. This process could be optimized grouping and compacting several turns in a single message [17,18]. Meanwhile, the recovering replica will be receiving turns corresponding to the regular replication messages, but their application has to be delayed until the ending of the recovery procedure. At that moment, the recovering replica will request for its inclusion in the *sec_sv*, which will trigger a subview change. If during the recovery procedure, the recoverer fails, a view change event will be fired, and the recovering replica will ask for a new one (lines 7-8). Note that some turns may have been already applied and do not need to be transferred again. This situation is prevented by line 10 in Algorithm 1, where *last_turn_recover* is updated. This variable will indicate an upper bound in the recovery transference.

5 Experimental Results

To verify the validity of our approach we performed some preliminary tests. We have implemented the proposed protocol on a middleware architecture called MADIS [15], taking advantage of its capabilities provided for database replication. For the experiments, we used a cluster of 4 workstations (openSUSE 10.2 with 2.6.18 kernel, Pentium4 3.4GHz, 2Gb main memory, 250Gb SATA disk) connected by a full duplex Fast Ethernet network. JGroups 2.1 is in charge of the group communication. PostgreSQL 8.1 was used as the underlying DBMS, which ensured SI level. The database consists of 10 tables each containing 10000 tuples. Each table contains the same schema: two integers, one being the primary key. Update transactions modify 5 consecutive tuples, randomly chosen from a table of the database. Read-only transactions retrieve the values from 1000 consecutive tuples, randomly chosen from a table of the database too. The PostgreSQL databases were configured to enforce the synchronization of write operations (enabling the fsync function). We used a load generator to simulate different types of workloads depending on the ratio of update transactions (10%, 50%, 90%). We simulated 12 clients submitting 500 transactions each one with no delay between them. The load generator established with each working replica the same number of connections than simulated clients. Transactions were generated and submitted through connections to replicas according to their role: update transactions to primary ones and read-only transactions to both primary and secondary ones. Note that, as these are preliminary tests, we have not paid much attention to the way transactions were distributed among the replicas and therefore results are not the best ones.

Experimental results are summarized in Figure 3. In the first two tests, we have tested the performance of our proposal working as a primary-copy and an update-everywhere approach respectively. Thus, starting from a primary replica (needed in both cases), we have increased the number of replicas depending on the evaluated approach: primaries for the update-everywhere operation and secondaries for the primary-copy one. As shown in Figure 3a, increasing the number of secondaries permits the system handling better read-only predominant loads (10% updates). However, in this primary-copy approach, it is impossible to enhance its performance when working with loads with a great number of updates (50% or 90% updates). In these cases, increasing the number of secondaries means no improvement, since additional secondaries do not increase the system capacity to process update transactions. In fact, all the update transactions are executed in the primary replica, and this overloads the replica.

On the other hand, the update-everywhere operation provides better results (see Figure 3b) than the primary-copy approach with loads including many update transactions (50% and 90% updates). In these cases, increasing the number of primaries allows to handle a greater number of update transactions and therefore the performance is improved. However, all the replicas are able to execute update transactions that may overload them and this may lead to higher response times when executing read-only transactions in these replicas. Besides, the coordination of the primary replicas involves also a greater overhead in their protocols than in a secondary protocol. For these reasons, the performance of the update-everywhere approach is poorer than the primary-copy one when the system works with a great number of read-only transactions.

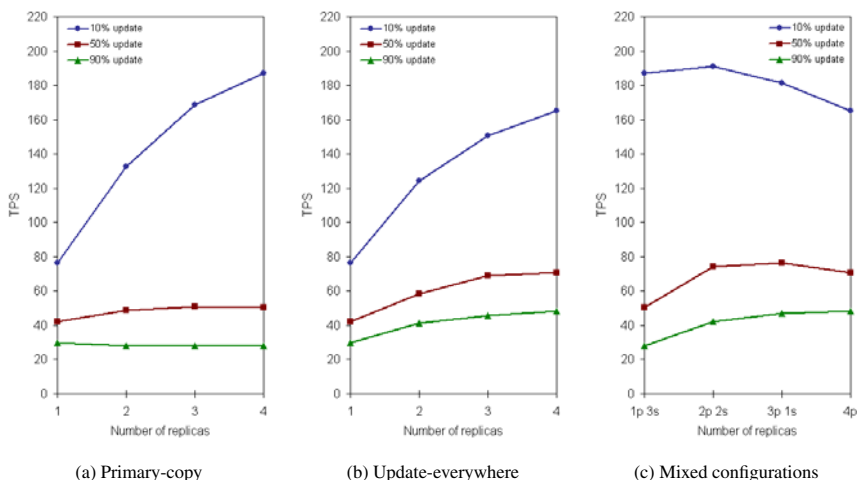


Fig. 3. Throughput for different analyzed workloads and configurations

We have seen that each approach behaves better under different loads. Hence, it is interesting to test how an intermediate approach (mixing several primaries and secondaries) performs. We have tested the behavior of mixed compositions, considering a fixed number of replicas. As shown in Figure 3c, mixed configurations with 4 replicas provide in general near the same and usually better results for each load considered in our tests. In particular, for a 10%-update load the best behavior (192TPS) is not provided by a pure primary-copy approach but by 2 primaries and 2 secondaries. This happens because using a single primary that concentrates all update transactions penalizes a bit the read-only transactions in such single primary replica, but with two primaries none of them gets enough update transactions for delaying read-only transaction service. Once again, for a 50%-update load the best throughput (76TPS) is given by 3 primaries and 1 secondary, outperforming a primary-copy configuration (51TPS) and an update-everywhere one (69TPS). This proves that intermediate configurations are able to improve the throughput achievable.

6 Conclusions

This paper has presented a new database replication approach, halfway between primary-copy and update-everywhere paradigms. The result is an improved performance, which is obtained since the protocol can change its configuration depending on the load. Moreover, it also allows to increase the fault-tolerance of primary-copy protocols. This is feasible thanks to the use of a deterministic database replication protocol that takes the best qualities from both certification and weak-voting approaches. This protocol establishes a unique schedule in all replicas based on primaries identifiers, which ensures that broadcast writesets are always going to be committed.

We have also discussed how this protocol can adapt itself dynamically to different environments (by turning secondaries into primaries to handle heavy-update workloads

or primaries into secondaries when read-only transactions become predominant). Finally, we have performed some preliminary experiments to prove the feasibility of this approach, showing how system can provide better performance adapting its configuration to the load characteristics, although we have still to make a great effort to achieve more significant results.

Acknowledgements. This work has been supported by the Spanish Government under research grant TIN2009-14460-C03.

References

1. Babaoğlu, Ö., Bartoli, A., Dini, G.: Enriched view synchrony: A programming paradigm for partitionable asynchronous distributed systems. *IEEE Trans. Comput.* 46(6), 642–658 (1997)
2. Berenson, H., Bernstein, P.A., Gray, J., Melton, J., O’Neil, E.J., O’Neil, P.E.: A critique of ANSI SQL isolation levels. In: *SIGMOD*, pp. 1–10. ACM Press, New York (1995)
3. Birman, K.P., Joseph, T.A.: Exploiting virtual synchrony in distributed systems. In: *SOSP*, pp. 123–138 (1987)
4. Chockler, G., Keidar, I., Vitenberg, R.: Group communication specifications: a comprehensive study. *ACM Comput. Surv.* 33(4), 427–469 (2001)
5. Daudjee, K., Salem, K.: Lazy database replication with snapshot isolation. In: *VLDB*, pp. 715–726. ACM, New York (2006)
6. Elnikety, S., Pedone, F., Zwaenopool, W.: Database replication using generalized snapshot isolation. In: *Symposium on Reliable Distributed Systems*, Orlando, FL, USA, pp. 73–84. IEEE-CS, Los Alamitos (2005)
7. de Mendivil, J.R.G., Armendáriz-Iñigo, J.E., Muñoz- Escof, F.D., Irún-Briz, L., Garitagoitia, J.R., Juárez-Rodríguez, J.R.: Nonblocking ROWA protocols implement GSI using SI replicas. Technical Report ITI-ITE- 07/10, Instituto Tecnológico de Informática (May 2007)
8. Gray, J., Helland, P., O’Neil, P.E., Shasha, D.: The dangers of replication and a solution. In: *SIGMOD Conference*, pp. 173–182. ACM, New York (1996)
9. PostgreSQL Global Development Group. PostgreSQL: The world’s most advanced open source database (2009), <http://www.postgresql.org/>
10. Jiménez-Peris, R., Patiño-Martínez, M., Alonso, G.: Non-intrusive, parallel recovery of replicated data. In: *IEEE Symposium on Reliable Distributed Systems*, vol. 0, p. 150 (2002)
11. Kemme, B., Alonso, G.: A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.* 25(3), 333–379 (2000)
12. Kemme, B., Bartoli, A., Babaoğlu, Ö.: Online reconfiguration in replicated databases based on group communication. In: *Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS), DSN 2001*, Washington, DC, USA, pp. 117–130. IEEE Computer Society, Los Alamitos (2001)
13. Kemme, B., Pedone, F., Alonso, G., Schiper, A., Wiesmann, M.: Using optimistic atomic broadcast in transaction processing systems. *IEEE TKDE* 15(4), 1018–1032 (2003)
14. Lin, Y., Kemme, B., Patiño-Martínez, M., Jiménez-Peris, R.: Middleware based data replication providing snapshot isolation. In: *SIGMOD Conference*, pp. 419–430. ACM, New York (2005)
15. Muñoz-Escof, F.D., Pla-Civera, J., Ruiz-Fuertes, M.I., Irún-Briz, L., Decker, H., Armendáriz-Iñigo, J.E., de Mendivil, J.R.G.: Managing transaction conflicts in iddleware-based database replication architectures. In: *SRDS*, pp. 401–410. IEEE-CS, Los Alamitos (2006)
16. Platner, C., Alonso, G., Özsu, M.T.: Extending DBMSs with satellite databases. *VLDB J.* 17(4), 657–682 (2008)

17. Ruiz-Fuertes, M.I., Pla-Civera, J., Armendáriz-Iñigo, J.E., González de Mendivil, J.R., Muñoz-Escobá, F.D.: Revisiting Certification-Based Replicated Database Recovery. In: Chung, S. (ed.) OTM 2007, Part I. LNCS, vol. 4803, pp. 489–504. Springer, Heidelberg (2007)
18. Vilaça, R.M.P., Pereira, J.O., Oliveira, R.C., Armendariz-Inigo, J.E., González de Mendivil, J.R.: On the cost of database clusters reconfiguration. In: Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems, SRDS 2009, Washington, DC, USA, pp. 259–267. IEEE Computer Society, Los Alamitos (2009)
19. Wiesmann, M., Schiper, A.: Comparison of database replication techniques based on total order broadcast. IEEE TKDE 17(4), 551–566 (2005)
20. Wu, S., Kemme, B.: Postgres-R(SI): Combining replica control with concurrency control based on snapshot isolation. In: ICDE, pp. 422–433. IEEE-CS, Los Alamitos (2005)

Educational Resource Scheduling Based on Socio-inspired Agents

Juan I. Cano^{1,2}, Eloy Anguiano^{2,3}, Estrella Pulido², and David Camacho^{2,*}

¹ Instituto de Ingeniería del Conocimiento, Spain

² Escuela Politécnica Superior - Universidad Autónoma de Madrid, Spain

³ Centro de Referencia Linux UAM-IBM, Spain

{inaki.cano, david.camacho, estrella.pulido
eloy.anguiano}@uam.es

Abstract. Scheduling a set of constrained resources is a difficult task, specially when there is no clear definition of ‘optimal’. When the constraints depend not only on physical or temporal issues but also in human desires or preferences the task gets harder. This is the case of educational resources, for example when a set of students must be distributed into a limited set of laboratories to attend to periodical practical sessions, in this case weekly. The preferences of the students may vary during the process for reasons such as the number of people already in that group. This paper presents a socio-inspired solution implemented as a multiagent system. The agents enroll themselves in the lab sessions based on their preferences and negotiate with other agents, using the resources they already have, to obtain desired groups that were already full.

Keywords: Scheduling, Multiagent system, Multiagent resource allocation, Constraint satisfaction problem, Socio-inspired, Complexity science.

1 Introduction

Resource allocation has always been a huge concern for administrators. These problems, from the family of constraint satisfaction problems, can be seen in several domains, as for example the allocation of processing time to the users of a mainframe [5] or the assignment of runways to planes in an airport [4]. Finding a way to solve these problems efficiently is important as some of them appear in time critical situations.

In general, in a resource allocation problem we have a set of resources that are limited and a set of agents that need these resources in some specific way. The nature and characteristics of these resources are very important when deciding a solution as they define part of the constraints to take into account. The rest of the constraints are defined by the agents or are imposed externally. A solution of the problem is found when we can make a feasible allocation (every agent has a resource assigned) or we find the optimal allocation. In the latter case we must decide on a way to measure the optimality of a solution. Also, there are cases where there is no possible solution.

* This work has been partially supported by the Spanish Ministry of Science and Innovation under grants TIN 2007-65989, TIN 2007-64718, and TIN 2010-19872.

This kind of problems have been treated in many different ways [3,7]. One approach recently developed is MultiAgent Resource Allocation (MARA)[2], which uses multi-agent systems to solve the allocation problem. This approach comes very natural as instead of programming an abstract algorithm we design a model of the problem, create some behavior for the agents and let system evolve to a solution. The key of this method is to capture the relevant aspects of the problem and define some utility function for the agents.

In the first section of this chapter there is a precise description of the problem, starting with an overview of the problem and then dealing with the relevant details to build the solution. In the second section we explain the design of the built system and why some decisions were made based on the description of the problem. Next, we will try to analyze the system from a complexity science point of view, identifying inputs, outputs, feedbacks and how they affect the system. Finally, we will show some results and conclusions, and how this system could be improved.

2 Description of the Problem

The problem to be solved was chosen for its familiarity and difficulty. Every year, in our university, the students enroll in some courses, usually five per term. The majority of this courses have two parts, one theoretical and one practical. The theoretical part of the courses are usually not a problem when allocating students, there are enough teachers and classrooms for the lectures, but for the practical part of the courses the laboratory space is limited. Even if there were huge computer labs, the relation between student and teacher is crucial for this part.

The practical sessions take place once a week. At the beginning of each term, the coordinator of the practical sessions for each course defines the number of groups, who is going to teach each of them and when the practical session for each group takes place. The group size is limited by space available in the lab. Students can join any of these groups, but once they choose a group they are required, unless there's a good reason, to attend to that group until the end of the term. This creates some conflict and competition among the students as they have time restrictions and preferences over the teachers (see fig. 1), and the group size limit makes that not every student is satisfied with her assigned groups.

This gets even more complicated as they have more than one course, usually five as mentioned earlier (see fig. 2). A student cannot join groups that overlap in time and this adds a dynamic restriction to the problem: once a student is assigned to a group, the time slot when the sessions take place are no longer available for other course's groups. This is a big problem from a CSP point of view, what once was a feasible solution or even an optimal solution can change after one assignment and become a terrible solution, and the other way around, what once was a terrible solution can become an optimal solution.

Once explained the problem in general terms, we can analyse it precisely to build a solution. In the following description we are going to use Resource Set (RS) for courses and Educational Resource (ER) for the groups. This way ER_1 will be the course number one and $RS_{1,1}$ will be the first group of the first course. To address correctly the

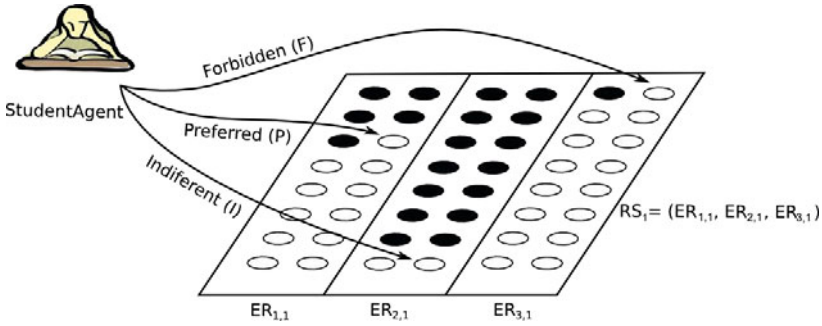


Fig. 1. Student preferences in a course with three groups. The whole square represent the course, the vertical divisions the groups. White dots are available places, black dots are occupied seats. The student in this figure prefers the first group, cannot assist to sessions in the third group and shows herself indifferent about the second group.

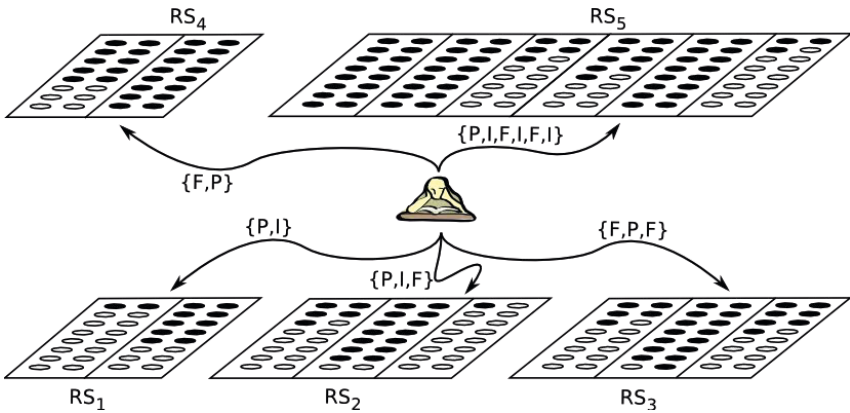


Fig. 2. A student’s global situation. Each RS (Resource Set) represents a course, and each course is divided as explained in fig. 1. The values between curly brackets represent the preferences of the student. For example, {P,I,F} means that the student prefers the first group (P), is indifferent about the second group (I) and cannot assist to the third group (F).

problem, it will be described by a set of properties and characteristics (resource types, preference representation, social welfare, allocation procedures, complexity, and simulation) from [2], which can be used to characterize MARA systems and applications.

– Resource Type

- *Discrete vs Continuous:* The ER are discrete, they cannot be represented with real numbers nor it can be divided.
- *Divisible or not:* The ER are not divisible.
- *Sharable or not:* Each ER can be assigned to more than one agent at the same time, but they have limits (seats/places).
- *Static or not:* The ERs doesn’t change during the negotiation, they’re not consumable nor perishable.

- *Single-unit vs Multi-unit*: Each ER is unique, it can't be confused with another ER.
- *Resource vs Task*: The ER is assigned as a resource, not as a task.
- Preference Representation
 - *Preference structure*: Although our model distinguishes between three different choices (Preferred (P), Indifferent (I), Forbidden (F)) of a particular educational item, and we could construct an ordinal preference structure (where $P > I > F$), we use an evaluation function that translates the agent preference into an integer which is used later to obtain a quantitative value, so a cardinal preference structure is the structure used.
 - *Quantitative preferences*: A utility function is used to map the bundle of resources assigned to an agent into a quantitative value, which will be later maximized.
 - *Ordinal preferences*: Not applicable.
- Social Welfare

Our approach is based on Collective Utility Function (CUF) because the aim is to maximize the average value of individual agent welfares. In the egalitarian social welfare, the aim is to improve the agent with the lowest welfare and in the utilitarian social welfare the aim is to improve the sum of all welfares, whereas in our approach the global state of the whole agent society by means of the average welfare is the aim of optimisation. Among the different possibilities (pareto optimality, collective utility function (CUF), leximin ordering, generalisations, normalised utility or envy-freeness) a CUF has been selected that defines the utilitarian social welfare as the total sum of agent cardinal values divided by the total number of agents.
- Allocation Procedures
 - *Centralized vs Distributed*: Our approach is fully distributed, since the solution is reached by means of a local negotiation amongst agents and there is not a global perspective of the ERA problem. An aggregation of individual preferences is used and the agent preferences are used to assess the quality of the global resource allocation.
 - *Auction protocols*: No auction algorithm is considered.
 - *Negotiation protocols*: A simplified version of the Concurrent Contract-Net Protocol (CCNP) has been implemented, where each agent can act as a manager and a bidder in the simulation step[9].
 - *Convergence properties*: Our negotiation algorithm needs a multilateral deal, where any interested agent in a particular educational item can negotiate with the manager (in our approach the agent who is trying to obtain a specific allocation).
- Complexity

Analysis of the models and assumptions, or the computational vs. communication complexity, used in our approach is not relevant for now.
- Simulation

A Multi-Agent Simulation Toolkit (MASON) has been used to deploy and test our proposed solution [6,8].

To summarize the restrictions of this problem, we have that each student must have one, and only one, group assigned for each course with practical sessions. The students

have preferences over the groups and are limited by other courses they are assisting, so the solution should maximize the student's happiness and can't assign to a student groups that overlaps in time. To these preferences we add that the groups must be evenly distributed and, by our institution requirement, courses from higher levels must have preference over the rest.

3 The Design of the System

The main objective of this work was to design a light system capable of solving this problem efficiently. Another objective was to use ideas and concepts that comes natural with the problem described. The multiagent approach let's the researcher use the problem terminology when constructing solution and the built system can be said to be socially inspired.

While building the solution, we created a model of how students interact between themselves when enrolling into groups. We observed that when signing into a group's list, students negotiate between them exchanging positions they have for the ones they want to have. This process is very local as they only contact friends and usually limit themselves to one subject or one group. In the model created, students negotiate with every other student that have something to offer. The exact negotiation process will be explained bellow.

Before explaining the negotiation, we need to understand how the students perceive their status and evaluate the proposals. We define the student's happiness or utility function as a function that increases with the number of groups assigned that they deem preferred. We have to maintain the groups balanced as this is good for the students as it is for the teachers, so the student happiness also varies with the occupation of the group in relation with the occupation of the other groups of this course. The happiness function would have the following form:

$$H(a_i) = \frac{\sum_{i=1}^n (f_1(RS_i(a)) + f_2(q, RS_i(a)))}{n}$$

In this equation we have $RS_i(a)$ that returns the group assigned to student a in the RS_i , q is the group's occupation, f_1 maps the student preference to an integer value, f_2 represents how the student perceive the balance of the group and n is the number of subjects the student has.

As explained in the above section, each student can assign one of three preference categories to each subject: P for Preferred, I for Indifferent and F for Forbidden. The values of f_1 for each preference category is 5 for P, 3 for I and 1 for F. We chose this values so that $P > I + F$ and F is better than not assigning a group.

The f_2 function assigns a value between 1 and 5 to the group occupation. The maximum value is achieved when the group occupation is exactly the same as the mean occupation of the groups, that is the number of students in a course divided by the number of groups that course has. To enroll in an empty group is always better than a full group, so full groups receive the minimum value. The shape of the f_2 function can be seen in fig. 3.

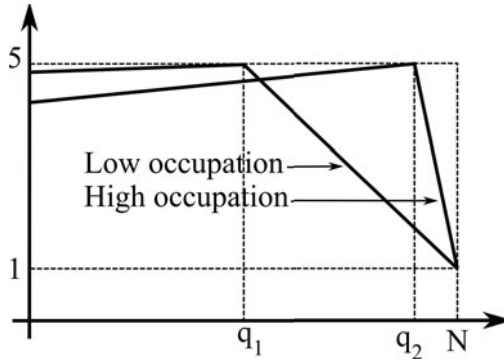


Fig. 3. Shape of the function that assigns a value to the occupation of a group. q_1 and q_2 are mean occupation of the course, N is the total number of students in the course. The slope of the function varies, depending on the number of students a course has.

This is the general version of the happiness function, during the execution of the system there are some variations. The first variation is that, to enforce the assignment of groups for the last year's courses (as required by the institution), when calculating the mean happiness this groups are valued double, we multiply the values of f_1 and f_2 by two. The other variation is that, when negotiating the students can ask once and again for the same group (we will see this later). To avoid this situation, each time a student cannot get a group in a negotiation, the happiness this group gives her will be reduced by a 10%.

The negotiation algorithm employed is a simplification of the Contract-Net Protocol [9]. The negotiation only takes place when a student wants to enter a group that is already full. In that case, the student interested in the group, let's call her the initiator, sends a list of groups she has already been assigned to to the students in that group, whom we will call the receivers. The list of groups sent only includes the groups that, if changed for the group the initiator is interested in, doesn't decrease her current level of happiness. The first receiver interested in an offered group, that is, his happiness is not reduced, swaps places with the initiator. A pseudocode of the algorithm can be seen below.

```
offer: = AssignedGroups(Initiator)
Filter(offer)

for all Receiver in Group do
  if Receiver is interested in $ER_x \in$ offer then
    Swap places
    return true
  end if
end for
return false
```

Before starting a negotiation, the student evaluates where would she be better. With the list of courses and groups she has assigned, she evaluates which group assignment can

be improved. If changing one group for another can be done, that is, the new group has enough space, the student swaps there directly. If the new group is already full, the student starts a negotiation with the students enrolled in that group. If there's no one to negotiate with or there's no group we can offer because any change would decrease the happiness, a desist factor is applied to that group. The happiness that group contributes to the student happiness is reduced by a 10% and at some point the student will start asking for a different group, possibly in a different course.

To find a solution for the problem, we let this model evolve until an equilibrium state is achieved. This equilibrium state is the state where no student wants to swap places with another student. The students start with no groups assigned and have complete freedom to enroll any group that is not full. As we stated before, MARA let's the designer use terminology and ideas from the problem to be solved with little abstractions needed.

4 Experimental Setup and Results

Once the system was built it needed to be tested. This section will detail the datasets employed and how the system built responded. To deepen in the study, the system was compared with a traditional CSP approach, as described in [1].

4.1 Data Sets

Several data sets, with incremental constraint-based complexity, have been considered. They have been generated by using real statistical information from the Escuela Politécnica Superior at Universidad Autónoma de Madrid (UAM). For each course, the number of laboratories available, students registered for each course, capacity of labs, and time tables, has been considered. This data has been used to generate a probability distribution of students/course and the number/capacity of labs that students need to attend. A four year degree has been considered (it corresponds to the current degree in Computer Engineering at UAM). Table 1 shows the student enrollment distributions by course (only those courses with laboratories are considered), the number of courses for which students have enrolled and the distribution of students with this number of courses.

Table 1. Distribution of labs by course

Year	No. Courses	Percentage of Students
1 st	2	100%
2 nd	2 - 3 - 4	20% - 50% - 30%
3 rd	4 - 5 - 6	15% - 70% - 15%
4 th	4 - 5 - 6	15% - 70% - 15%

Table 1 assumes that students from any year has at least one course from that year and no courses from higher years. This means that a third year student has at least one third year course and no fourth year courses. This can also mean that a third year students

can be enrolled in first and second year courses. To simplify the tests, we limited this so that a student can only have courses from one year and the immediately below. The first and second year has a total number of 2 courses with practical sessions, we don't take into account theoretical-only courses, while third and fourth year have 5 courses with practical sessions.

As an example of how to read the table, the second row represents the second year students. This students can have 2, 3 or 4 courses, and some of them can be from the first year, up to 2 as there are only 2 courses in the first year. From the total number of second year students, a 20% have 2 courses, 50% have 3 courses and 30% have 4 courses.

As mentioned above, students can have courses from one year below the current year. In the second year, students with only 2 courses have both from second year, students with 3 have 1 course from first year and students with 4 have 2 from first year. In Table 2 the fraction of courses from another year are shown for the third and fourth years. The table shows the distribution of courses of different years for each student. By examining the real data, it can be noted that the number of students with at least one course from another year is higher than the number of students that have courses only of their own year. In addition, the number students with 6 courses of the same year are very low.

Table 2. Distribution of labs for 3rd and 4th year courses

No. enrolled courses	All the same year (%)	One course from one year back (%)	Two courses from one year back (%)	Three courses from one (%)
4	20	60	20	0
5	10	40	50	0
6	5	40	40	15

Based on the previous information, six data sets of 1000 students were generated. The synthetic restrictions for each student were also randomly generated by using some historic data related to previous years. Table 3 summarizes the basic features for each data set, where Ds0 considers only one Preferred group per ER and StudentAgent (e.g. $\langle\langle P, I, I \rangle, \langle P, I, I \rangle\rangle$), whereas Ds5 considers that 30% of ERs are marked as Preferred and the rest (70%) are Forbidden. For example, $\langle\langle P, F, F \rangle, \langle P, F, F \rangle, \langle P, F, F \rangle\rangle$ makes a 30–70 distribution. The number of Forbidden and Preferred constraints has been adjusted along different datasets to cover different complexity situations.

The difference between the percentage of P and F is the percentage of I. This way, the most restrictive datasets are DS6 and DS7 because the number of I is reduced and in some cases, many cases, there will be no I in the student preferences.

4.2 Results

Two systems were tested using these datasets, a CSP [1] and the multiagent system described in the previous section. Tables 4 and 5 show the results obtained for both systems. Since the happiness functions were obtained by using different preference values,

Table 3. Student datasets

Data Set	Preferred groups (P)	Forbidden groups (F)
Test	100%	0%
Ds0	1P/(RS,agent)	0%
Ds1	30%	0%
Ds2	1P/(RS,agent)	20%
Ds3	30%	20%
Ds4	1P/(RS,agent)	50%
Ds5	30%	50%
Ds6	1P/(RS,agent)	70%
Ds7	30%	70%

Table 4. CSP experimental results for datasets considered

Data Set	Mean Happiness	Happiness Deviation	Mean Distribution	Distribution Deviation	P's (%)	I's (%)	F's (%)
Test	9.85	0.07	81.5	0.66	100	0	0
Ds0	9.69	0.15	81.3	21.8	86.9	13.1	0
Ds1	9.78	0.1	81.2	15.2	94.2	5.8	0
Ds2	9.32	0.25	81	17.7	89.8	1.07	9.19
Ds5	7.82	1.09	80.6	17.9	61.3	15.6	23.1
Ds6	7.03	1.12	80.8	15.8	66	8.2	25.8
Ds7	6.02	1.59	80.9	19.5	71.9	0	28.1

they are not comparable across systems but they give a good estimation of their performance with different restrictions. The key values for the comparison are the percentage of P, I, and F in the final allocation.

As seen in Table 4, the CSP gives a good result for the first datasets, but when the restrictions are increased the overall happiness decreases and the percentage of F assigned is very high. Although the execution time was not registered, it was comparatively high when comparing with the MARA approach. While MARA took only a few seconds, the CSP approach need hours to do the assignment. For datasets 3 and 4, it took so long that it was stopped before completion, after 6 hours of execution.

The results obtained for the MAS model are shown in Table 5. This method is able, by using the negotiation-based approach, to maintain the global happiness of the solutions found. Although “happiness” values cannot be directly compared between CSP and MAS solutions (because their equations are different), the percentage of allocated F can be compared. In the worst situation (DS7) only the 4% of the students needs to be assigned to a forbidden ER (F), and the 96% of student teams are satisfactory allocated. Finally, the low variation of the happiness among the different datasets can be remarked compared to the variation of this value for the CSP solutions. This is due to the facility (given by the Multi-agent approach) to change preference values, or to exchange the current ER allocation with other agent in the system.

Table 5. MAS experimental results for datasets considered

Data Set	Mean Happiness	Happiness Deviation	Mean Distribution	Distribution Deviation	P's (%)	I's (%)	F's (%)
Test	9.85	0.07	81.5	0,66	100	0	0
Ds0	9.3	0.29	81.3	4.21	85.3	14.7	0
Ds1	9.7	0.14	81.4	0.64	94	6	0
Ds2	9.4	0.24	81.1	4.36	86.9	12.75	0.35
Ds3	9.8	0.12	80.4	2.04	95.8	3.67	0.49
Ds4	9.2	0.33	80.8	6.94	85.7	12.5	1.8
Ds5	9.7	0.19	81.0	0.89	95.5	2.42	2.05
Ds6	9.1	0.43	81.0	5.92	86.0	11.1	2.9
Ds7	9.7	0.29	80.9	0.91	95.8	0	4.2

5 A Complexity Science View

Although there is no agreement on a definition of complex, we will use its etymological definition. The word comes from latin *complexus*: *com-* (“together”) and *plectere* (“to weave, braid”). If we refer to its etymology, something complex should have at least two elements and have some intricate relationship between its elements. Thus, a complex system should be a system with different elements and a mesh of relationships among them.

In this system we can identify two sources of complexity. On one hand we have the problem itself. The courses and groups are related to each other, and also are the students. The relationships between the different parts of the problem makes it a complex environment and if we introduce some change in it, e.g. change a group’s time slot or the student preference, the reaction is unpredictable as we can’t determine how other students will react or how the constraints will change during a simulation, among other things.

On the other hand we have the built system. As cyberneticists say, the best way to deal with a complex system is with another complex system. The built system is completely based on the problem, each part of it is a model of some aspect or aspects of the problem. This makes the built system to be adaptive and flexible: after a solution is found, we can translate any change in the original environment into the system and let it find a new solution. This adaptivity and flexibility is not available in other methods as, for example, backtracking. If we want to introduce a change in the problem, we would need to run the backtracking algorithm from the beginning because some branches that were not useful before can lead to solutions now.

The built system is composed of different agents, each one of them can be differentiated from the others by the courses they have and the preferences over the groups in each course. The interaction between the agents is local, they can only communicate with a limited part of the system. At most, an agent can establish communication with other agents in the same courses it has, but normally they will communicate with only a subset of these agents. Although this communication is local, we have a global situation that the agents are not aware of.

The outcome of the system, the stable state, cannot be traced back to its agents. We can say that agent X enrolled in a group she can't attend, i.e. has an F for that group in her preference vector, because the group she wanted was full and no one there is interested in other groups she has. But then we need to go back and see why nobody is interested or why she couldn't enter the group in the first place because everyone had an equal chance to get into it. This causal chain is long, complicated and full of suppositions and, although we could find some probabilities over the links, we cannot fully explain why the resulting state is the one it is.

Complex self-organizing systems have feedback mechanisms. Feedback mechanisms defines how outputs are related to inputs, positive feedback increases the effect of an input while negative feedback reduces this effect. Before defining inputs and outputs we must define where are the borders of the system, in other words, what is part of the system and what is part of the environment. As we don't expect changes in the courses or groups, we define the system as the group of agents and the environment as the courses and groups. This way we have that the problem defines an environment with the following variables:

- List of courses
- Number of groups for each course
- Size of the groups
- Groups' hours
- Number of agents
- Enrolled courses for each agent
- Preference vectors for each agent

This variables can be understood as how the agent models its environment and its goals, they belong to both, the environment and the system, but they are not fed into the system nor they are extracted from it. Previously we discussed the adaptivity of this kind of systems and how we could change the environment while the system is running. This stays true after this separation, what changes is how we introduce these modifications.

The rest of variables depends directly on the agents and they can modify them as they wish. From this variables we will define as an output, i.e. variables that the system shows to its environment, the assigned groups for each agent, and the only input would be the distribution of students in groups. To define the state of an agent we use the list assigned groups, the number of time it has attempted to enter a group and its happiness.

To recapitulate, we have that the problem is defined by variables that determine the courses and groups and their relationship with the agents. We can consider this variables as immutable and part of the agents' internal representation of the world. The list of groups assigned, the number of times an agent has attempted to enter a group and its happiness are the only variables necessary to determine the agents state. The system constantly receives information about the state of the groups, meaning that it knows what students are already in a group or if a group is full. In exchange, the system informs about changes it makes in the groups.

We can see clearly that there is a feedback loop, the state of the groups is fed to the agents and the agents inform about any changes in them. This loop doesn't determine by itself if it is a positive or negative feedback, it depends on how the system treats this

information. In our case, this loop is managed in two ways: with the insistence factor and the f_2 function, both in the happiness equation.

The insistence factor is applied when the agents try to enter any group more than once without any success. The relation with the feedback is a little difficult to see, first the agent receives the status of the groups for the courses it has enrolled. The agent evaluates if there is a group that would increase its happiness and, if the group is full, it tries to negotiate with other student to enter. It is possible that the list of groups an agent receives doesn't change from one attempt to the next and the agent finds that the group that improves its happiness is the same full group each turn. To dissuade the agents, the insistence factor is applied. This makes the feedback loop negative as it turns the output to be, after some time, the same as the input, whatever the input is, and the agents to stop trying to change groups.

For the other way to manage feedback, the f_2 function, it is clearer that it affects the feedback, but whether it does in a positive or negative way is hard to decide. When a group is below what was defined as the optimal occupancy this function encourages the agents to enroll this group. Once this quota is filled, the function dissuades agents from entering and sending them to other groups that are less full by reducing very fast the happiness this group can provide.

The value for the insistence factor and the shape of the f_2 function must be chosen carefully and with a goal in mind. These two mechanisms reduce the agent activity and drive the system to a stable state. The insistence factor is negative in nature as it forces the students to stop their activity. The f_2 can have a positive and negative effect, depending if the occupancy of the group is below the optimal or over it. This function also increases the variety of options, as agents do not only search for preferred group but also for groups that are not overcrowded.

As we can see, the built system is complex and self-organizes. This gives great flexibility when finding a solution, but makes the system hard to control. By introducing mechanisms to control the feedback we can ensure that the system will come to an equilibrium where the agents would stop searching, but a fine tuning of the system or proving that it will find the optimal solution are very difficult tasks if not impossible.

6 Conclusions

Scheduling problems are always difficult to solve. The elements of the problems usually have intricate relations between them and usually, as in our case study, these relations make the restrictions vary during the process of solution. Traditional approaches as backtracking and related prove to be very accurate finding the solutions, but the cost in time and computational power is very high. Also, they are not easily comprehensible and cannot adapt to changes in the initial conditions.

The use of multiagent systems to solve this kind of problems overcomes these obstacles, as the agents can be simple computationally speaking and their behavior can be easily explained using the problem terminology. The concurrent and distributed nature of multiagent systems makes them very robust and adaptive.

The problem with the multiagent approach is the design process, as small perturbations in the agents' behavior can result in very different behaviors of the system as a whole. One way to deal with this is to design the system as a model of some natural or

social system that solves the same problem or a similar one. In this case, the agents are modelled as the students of the university that negotiate with each other for the groups they want.

The models considers three characteristics of the students: their preferences, the search for a group that is not empty nor full and the fatigue from trying to enter a group without success. With these three characteristics included in the model, the results are very good in comparison with an implementation of a backtracking algorithm.

For a successful implementation of a system like the one presented, it is needed to understand the roles of positive and negative feedback. Positive feedback makes a system more controllable but unpredictable, because small inputs can render big changes. The system can be moved to another state, but the new state is not predictable. On the other hand, negative feedback makes the system uncontrollable but predictable as even big inputs cannot change the system state. A balance between these two feedback mechanisms can drive a system to self-organize.

To sum up, when trying to solve a complex problem it is useful to adopt ideas from natural and social systems that solve similar problems. The use of multiagent systems is highly recommendable as they can be, if well constructed, resilient and adaptive. Taking into account the interplay of positive and negative feedback in the design process is important for self-organization, giving independence to the agents and not considering this mechanisms can lead to disastrous outcomes as the system can never get to a stable state or get there too soon.

References

1. Cano, J.I., Sánchez, L., Camacho, D., Pulido, E., Anguiano, E.: Allocation of educational resources through happiness maximization. In: Proceedings of the 4th International Conference on Software and Data Technologies (2009)
2. Chevaleyre, Y., Dunne, P.E., Endriss, U., Lang, J., Lemaître, M., Maudet, N., Padget, J., Phelps, S., Rodríguez-Aguilar, J.A., Sousa, P.: Issues in multiagent resource allocation. Special Issue: Hot Topics in European Agent Research II Guest Editors: Andrea Omicini 30 (2006)
3. Choueiry, B., Faltings, B., Noubir, G.: Abstraction methods for resource allocation. Tech. Rep. TR-94/47, Département d'informatique, Institut d'informatique fondamentale IIF (Laboratoire d'intelligence artificielle LIA) (1994)
4. Gilbo, E.P.: Optimizing airport capacity utilization in air traffic flow management subject to constraints at arrival and departure fixes. *IEEE Transactions on Control Systems Technology* 5(5) (1997)
5. Jain, R., Chiu, D., Hawe, W.: A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. DEC Research Report TR-301 (1984)
6. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K.: Mason: A new multi-agent simulation toolkit. In: Proceedings of the 2004 Swarm Fest Workshop (2004)
7. Modi, P., Jung, H., Shen, W., Tambe, M., Kulkarni, S.: A dynamic distributed constraint satisfaction approach to resource allocation. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, p. 685. Springer, Heidelberg (2001)
8. Railsback, S.F., Lytinen, S.L., Jackson, S.K.: Agent-based simulation platforms: review and development recommendations. *Simulation* 82(9) (2006)
9. Smith, R.G.: The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers* (1980)

Part IV

Data Management

Managing Risks by Integrity Constraints and Integrity Checking

Hendrik Decker*

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia
Ciudad Politécnica de la Innovación, c. Vera 8G, 46022 Valencia, Spain
<http://web.iti.upv.es/~hendrik/>

Abstract. Semantic properties of any kind, and in particular the riskiness of data can be modeled and monitored by conventional database integrity technology. As opposed to conventional integrity constraints, occasional violations of some of the constraints that capture risk aspects may be tolerable, even for extended periods of time. Traditional integrity checking methods are intolerant wrt. any constraint violation. They insist that all constraints are totally satisfied before updates can be checked for integrity preservation. Inconsistency-tolerant methods can waive that insistence. Thus, if risks are modeled by constraints, they can be monitored by any integrity checking method that is inconsistency-tolerant. We illustrate that by an extended example, in which our inconsistency-tolerant solution is also compared to some alternative approaches.

Keywords: Risk management, Constraints, Integrity checking, Inconsistency tolerance.

1 Introduction

In relational databases, knowledge bases and decision support systems, first-order predicate logic sentences called *assertions* or, more specifically, *integrity constraints*, or simply *constraints*, are used to express conditions that are required to be invariantly satisfied across state changes caused by updates.

The expressive power of logic also can be used to capture any other semantic information that goes beyond the simple structures of common database content. For example, the quality of data can be modeled by sentences that have the form of integrity constraints, and then measured, monitored and maintained by methods that otherwise are used for integrity checking, as shown in [5].

Similarly, conditions for characterising risks associated to any state of the database can also be expressed in the syntax of constraints, and monitored by integrity checking methods, as we are going to see in this paper.

The basic idea of managing risks by constraints is that the stored data can be considered to be risk-free if the database satisfies all constraints that capture conditions indicating risks. More generally, the number of violated instances of constraints can be

* Partially supported by EU ERDF and the Spanish grants TIN2009-14460-C03, TIN2010-17193.

considered as a measure of the extant risk as reflected by the current contents of the database. If a critical amount of constraint violations is surpassed, then corresponding risks have increased beyond tolerable proportions. Each update that would increase the amount of extant risk beyond a certain threshold should be rejected.

In other words, violations of constraints that capture low risks may be tolerable, while the avoidance of violations of constraints that capture high risks should have higher priority. Thus, the higher the risk associated to a constraint (or, more precisely, the higher the risk expressed by a particular violation of a constraint), the lower should be the tolerance with which such violations are met by the integrity maintenance module of the database system.

Usually, constraints are sentences that involve universal quantifications of variables in several relations with conditions that correspond to possibly huge joins of large tables. Thus, the evaluation of constraints tends to be prohibitively expensive. Hence, efficient methods for simplifying the evaluation of integrity constraints are needed. However, since the intended semantics of risk constraints may be different from that of integrity constraints, the use of integrity checking methods for simplifying the evaluation of assertions about risks is questionable.

Traditional integrity checking insists on total constraint satisfaction. However, that is not suitable in general for monitoring risks constraints, since some of them may be occasionally violated, even for extended periods of time, without impairing ongoing routine operations. As opposed to that, we are going to see that integrity checking methods that are able to tolerate extant violations of constraints also are able to monitor the dynamics of risky data.

More precisely, we show how to gain a better control over the riskiness and possible imperfections of stored data, by expressing risk properties as constraints, and monitoring them with inconsistency-tolerant integrity checking methods.

Conditions that model risk properties may either qualify data positively, e.g., as trustworthy, secure or robust, or negatively, e.g., as imperfect, uncertain, vague, corrupt, out of range, etc. In general, positively stated properties assert the absence of risk, while negatively stated properties assert the presence of risk. In both cases, a risk constraint is said to be satisfied if the truth value of the property it describes implies that no risk associated to that property has to be feared. Conversely, a risk constraint is violated if its truth value entails that some risk is manifest in the data.

Capturing risk properties of data by describing them in the form of integrity constraints yields a double benefit. Firstly, it shows that the expressive power of the syntax of semantic constraints can indeed be used also for modeling more general properties of data, such as their risk potential. Secondly, it enables the use of established integrity checking methods in order to efficiently monitor the risk potential of stored and incoming new data.

In section 2, we first strive to gain a better understanding of the similarities and differences between constraints for integrity and risk. Then, we claim that, in spite of seemingly severe differences, it is possible to capture risk conditions by integrity constraints, and to monitor them by using methods for integrity checking. This claim is substantiated in the remainder of the paper. In section 3, we recapitulate the concept of inconsistency-tolerant integrity checking [4,6]. Inconsistency here is synonymous to

integrity violation. We show that it is precisely the inconsistency tolerance of methods that makes them apt to be used for monitoring risks. In section 4, we elaborate an extended example that illustrates how inconsistency-tolerant integrity checking can be used for risk management. It will become obvious that risk management is a special case of managing the quality of data. In section 5, we address related work. In section 6, we conclude.

2 Risk and Integrity

In 2.1, we analyse the similarities and in 2.2 the differences between constraints that either model risk or integrity. By several examples, we illustrate that, syntactically, conditions for integrity and for risk are very similar. Semantically, they differ since integrity constraints and their evaluation traditionally are much more exigent than assertions about risks. As explained in 2.3, this difference is reconciled and can be overcome by the inconsistency tolerance of methods for integrity checking. With such methods, constraints can be evaluated upon updates even if some instances of the constraints are violated already before the update.

2.1 Similarities

Traditionally, integrity constraints are used to express correctness conditions with which all stored data must comply in each state of the database. Upon each issued update, the constraints imposed on the database are checked. Updates are committed only if they do not cause integrity violation. For example, in a hospital database, the integrity constraint

$$\forall x \forall y \forall z (\text{glycohemoglobin}(x, y, z) \rightarrow \text{person}(x) \wedge \text{percent}(y) \wedge \text{datetime}(z))$$

requires that the first attribute of the *glycohemoglobin* table always is a person, the second a percentage value and the third of type *datetime*. Thus, inserting

$$\text{glycohemoglobin}(\text{john}, \text{'low'}, \text{2010-09-27.11:15})$$

into the database would violate its integrity, since the value *'low'* is ruled out by the constraint. Similarly, the denial

$$\leftarrow \text{glycohemoglobin}(x, y_1, z), \text{glycohemoglobin}(x, y_2, z), y_1 \neq y_2$$

imposes a primary key constraint on the combination of the first and the third column, thus preventing duplicate entries for the same person with different glycohemoglobin levels at the same time.

Also conditions for characterising database entries that are risky can be modeled in the syntax of integrity constraints.

For instance, consider the denial constraint

$$\leftarrow \text{risk}(x, z)$$

where the predicate *risk* is defined by

$$\text{risk}(x, z) \leftarrow \text{person}(x), \text{glycohemoglobin}(x, y, z), \text{above-threshold}(y)$$

and *above-threshold*(*y*) is defined by comparing the value of *y* with a suitable constant. This constraint qualifies the well-being of each person *x* to be at risk at time *z* if the glycohemoglobin level *y* is above a permissible threshold value.

Another example of a risk constraint is

$$\leftarrow \text{birth-date}(x, z), z < 1900$$

by which each entry of persons x in the register of living persons of some municipal administration with birth date z before the 20th century can be characterised as dubious and thus risky, in terms of trustability.

By amalgamating higher-order predicates into first-order terms [3], also sentences such as

$$\text{confidence}(\text{row}(x, y), z) \wedge z < th \rightarrow \sim \text{trustworthy}(x)$$

may serve as constraints for indicating risks associated to rows x in database tables y such that the confidence value z of x is below a certain threshold value th .

So, we have seen that it is possible to characterize risky data by the syntax of integrity constraints. Hence, it suggests itself that it could as well be possible to use integrity checking methods in order to check tuples that are requested to be inserted or deleted for violations of risk constraints. In the following subsection, we are going to see that this is not as straightforward as it may seem at first thought.

2.2 Differences

In 2.1, we have seen that the representation of properties describing the logical consistency of stored data or their risk potential is very similar. Both can be modeled by integrity constraints.

However, there is a significant difference between risk and integrity violation. Data that lack integrity are not just risky, but definitely bad, while risky data may or may not have integrity. Essentially, the difference is that integrity is two-valued (i.e., satisfied or violated), while risk is not binary, no matter if risk is defined qualitatively or quantitatively. In other words, risky data may be imperfect and may have an impaired quality, but they are not necessarily invalid or inconsistent. In fact, data that violate integrity are usually considered detrimental, harmful and unwanted, while data that are risky typically may carry useful information. For example, the integrity constraint

$$\text{violated} \leftarrow \text{emp}(x), \text{age}(x, y), y < 14$$

expresses that integrity is violated by underage employment (because there is a law by which this constraint is enforced), while the assertion

$$\text{risky} \leftarrow \text{emp}(x), \text{age}(x, y), y > \text{retirement_age}$$

expresses that overage employment qualifies as risky. Here, riskiness may be interpreted in various ways. One way could be that the correctness of the information captured by the fact $\text{age}(x, y)$ should be considered doubtful (since it may contradict an employer's general policy that a person beyond retirement age would remain employed), or problematic in terms of health considerations, or because of a labour legislation that rules out most cases of overage employment.

Another example for illustrating the semantic difference between integrity and risk is, on one hand, the integrity constraint

$$\text{violated} \leftarrow \text{email}(x), \text{sent}(x, y), \text{received}(x, z), y > z$$

which declares that integrity is violated if the sent-date of an email item is after its received-date (assuming that both x and y are normalised wrt the same time zone). On the other hand, the formula

$$\text{suspect}(x) \leftarrow \text{email}(x, \text{from}(y)), \sim \text{authenticated}(y)$$

rates an email item x received from y as suspect if the latter cannot be authenticated, although the message content of x may well be valid and unproblematic.

Although essentially the same syntax can be used to represent conditions for integrity and (lack of) risk, the use of known integrity checking methods for monitoring the riskiness of data must be deemed questionable, if not unfeasible, for the following reason.

All methods for efficient integrity checking insist that integrity must be satisfied before a given update is checked for integrity. That way, the evaluation of constraints can focus on the relevant part of the data that are actually affected by the update, while the rest can be ignored, since it is known to satisfy integrity. In general, it would be unrealistic, however, to always insist on total integrity satisfaction by requiring that, before each update, all stored data should comply perfectly with all requirements imposed by constraints. After all, certain defects of quality and risks associated to the data can never be excluded with complete certainty. Thus, not all data can be assumed to be perfectly risk-free whenever an update needs to be checked for introducing higher risks. Examples of data the reliability of which is risky are given by the contents of each large thesaurus or encyclopedia (think, e.g., of Wiktionary or Wikipedia).

In principle, a way out of this dilemma could be to use a method that does not insist on total integrity before each update. The only method in the literature that does not require the total satisfaction of all constraints is the so-called *brute-force* method. It exhaustively evaluates all constraints upon each update, without any simplification. But brute-force evaluation may be prohibitively expensive, due to the high complexity of constraints. Another way out could be to repair all violated constraints before or after each update.

A more elegant and less expensive solution of using integrity checking methods for monitoring risk constraints is presented in the following section.

2.3 Reconciliation

Inconsistency-tolerant integrity checking has been discussed in [4,6]. In particular, it has been shown that, contrary to common belief, many well-known integrity checking methods, although not all of them, can waive the requirement that each consistency constraint be totally satisfied before updates can be checked efficiently for integrity preservation. An important feature of each inconsistency-tolerant method is that none of its functionality and efficiency is compromised by arbitrarily high amounts of extant constraint violations.

Since risk-related properties can be expressed by the syntax of integrity constraints, it follows that inconsistency-tolerant integrity checking methods can be used to also monitor such properties. In particular, the use of inconsistency-tolerant methods enables an efficient way of evaluating risk constraints even if there are data that do not fully comply with all of them.

Nevertheless, inconsistency-tolerant methods are capable of detecting and rejecting each impairment of risk constraints upon each update, no matter if the extant

infringements of such assertions are minor shortcomings, serious risk indicators or even major corruptions of data. Thus, the task of reducing risks (or, more generally, of improving the quality of damaged data) can be delegated to separate, possibly off-line processes. Such processes may be run at any convenient point of time. In particular, they need not be run at update time, as required by traditional integrity checking approaches.

3 Inconsistency Tolerance

In this section, we recap the main definitions of inconsistency-tolerant integrity checking [4,6]. Unless specified otherwise, we use terminology and notations that are conventional in the databases community (see, e.g., [12]).

Throughout, let ‘method’ always signify an integrity checking method. We assume that each constraint is represented in *prenex form*, i.e., an implicit or explicit quantifier precedes a quantifier-free matrix. This includes the two most common forms of representing a constraint, either as a denial (i.e., a clause without head whose body is a conjunction of literals) or in *prenex normal form* (i.e., quantifiers outermost, negations innermost). An *integrity theory* is a set of constraints. An *update* is a bipartite finite set of database clauses to be inserted or deleted.

From now on, let the symbols D , IC , U , I and \mathcal{M} always denote a database, an integrity theory, an update, a constraint and, resp., a method. For each update U , we write D^U to denote the updated database, and also refer to D and D^U as the *old* and the *new* state, respectively.

We assume that the semantics of D and IC is given by a distinguished unique Herbrand model of D . Thus, I is *satisfied* (*violated*) in D if I is *true* (resp., *false*) in that model. As usual, IC is called *satisfied* (*violated*) in D if each $I \in IC$ (resp., at least one $I \in IC$) is satisfied (resp., violated) in D . For convenience, we write $D(IC) = true$ and $D(I) = true$ for denoting that IC or, resp., I is satisfied in D , and $D(IC) = false$ ($D(I) = false$) that it is violated. ‘Consistency’ and ‘inconsistency’ are synonymous with ‘satisfied’ and, resp., ‘violated’ integrity.

Each correct method \mathcal{M} can be formalized as a mapping that takes as input a triple (D, IC, U) such that $D(IC) = true$, and outputs upon termination either *ok* or *ko*. Here, *ok* means that \mathcal{M} accepts U because U does not violate any constraint, and *ko* means that \mathcal{M} does not accept U . For inconsistency-tolerant methods, the premise $D(IC) = true$ can be waived without penalty. For simplicity, we only consider input triples (D, IC, U) such that the computation of $\mathcal{M}(D, IC, U)$ terminates. In practice, that can always be achieved by a timeout mechanism with output *ko*.

Each constraint I can be conceived as a set of particular instances, called ‘cases’, of I , such that I is satisfied if and only if all of its cases are satisfied. Thus, integrity maintenance can focus on cases, and check if their satisfaction is preserved across updates. Violated cases can thus be temporarily tolerated and possibly be repaired at any convenient moment. That is captured by the following definition.

Definition (*Inconsistency-tolerant Integrity*).

a) A variable x is called a *global variable* in I if x is \forall -quantified in I and \exists does not occur left of the quantifier of x .

b) For a constraint I and a substitution ζ of its global variables, let $I\zeta$ be obtained by replacing each global variable in I by the term assigned to it in ζ . Each such $I\zeta$ is called a *case* of I .

c) Let $\mathbf{SC}(D, IC)$ denote the set of all cases C of all $I \in IC$ such that $D(C) = \text{true}$, i.e., C is satisfied in D .

d) \mathcal{M} is called *inconsistency-tolerant* if, for each triple (D, IC, U) , the output $\mathcal{M}(D, IC, U) = \text{ok}$ entails that $D^U(C) = \text{true}$, for each $C \in \mathbf{SC}(D, IC)$.

In words, the definition above means: If an inconsistency-tolerant \mathcal{M} accepts an update without insisting that each constraint be satisfied before the update, then the output *ok* guarantees that each case of IC that was satisfied in D remains satisfied in D^U .

Example. For relations p, q , let the second column of q be subject to the foreign key constraint $I = \forall x, y \exists z (q(x, y) \rightarrow p(y, z))$, which references the primary key column of p , constrained by $I' = \leftarrow p(x, y), p(x, z), y \neq z$. The global variables of I are x and y ; all variables of I' are global. For $U = \text{insert } q(a, b)$, a typical method \mathcal{M} only evaluates the simplified basic case $\exists z p(b, z)$ of I . If, for instance, (b, b) and (b, c) are rows in p , \mathcal{M} outputs *ok*, ignoring all irrelevant violated cases such as, e.g., $\leftarrow p(b, b), p(b, c), b \neq c$ and I' , i.e., all extant violations of the primary key constraint. \mathcal{M} is inconsistency-tolerant if it always ignores irrelevant violations. \mathcal{M} outputs *ko* if there is no tuple matching (b, z) in p .

It is easy to see that inconsistency-tolerant integrity checking significantly generalizes the traditional approach, which does not legitimize the use of methods in the presence of extant constraint violations.

As shown in [4,6], many known methods for integrity checking are inconsistency-tolerant. The reasoning of such methods, and also of methods that are not inconsistency-tolerant, is featured in section 4.

4 Risk Management

In this section, we illustrate how to use the evaluation of assertions by integrity checking methods for managing risks.

A risk is a negative quality. Its positive counterpart may be characterized by properties related to security, dependability, reliability, safety and the like. Risks can often not be totally excluded, while it is always requisite to minimize and to control them for lowering their probability to increase.

In general, the amount of tolerable risk depends on the application and often also on its users (think, e.g., of stock market transactions). The example elaborated below is open to interpretation. By assigning convenient meanings to predicates, it could be interpreted as a risk model of, e.g., financial services (think, e.g., of Basel II), or a nuclear power plant.

Of course, a single example can always be criticized to be statistically irrelevant. However, for each of the mentioned alternatives, several typical features that are independent of the particular example are illustrated. In particular, we are going to see that, for safety-critical applications, the use of a method that is inconsistency-tolerant

is more dependable than to use one which is not. Our example will show that using a non-inconsistency-tolerant method for monitoring risks may have fatal consequences.

We are going to compare inconsistency-tolerant integrity checking with the following alternative approaches to monitor risk: brute-force evaluation, non-inconsistency-tolerant integrity checking, repairing, and consistent query answering [1]. In detail, we address the following points 1) - 6), in subsections 4.1 - 4.6.

- 1) The cost of the brute-force method.
- 2) The cost of inconsistency-tolerant methods.
- 3) The dependability of methods.
- 4) The cost of repairing the old state.
- 5) The cost of repairing the new state.
- 6) The risk of consistency query answering.

Let us consider a database D with the following definitions of view predicates rl , rm , rh that model risks of low, medium and, respectively, high degree.

$$\begin{aligned} rl(x) &\leftarrow p(x, x) \\ rm(y) &\leftarrow q(x, y), \sim p(y, x) \\ rm(y) &\leftarrow p(x, y), q(y, z), \sim p(y, z), \sim q(z, x) \\ rh(z) &\leftarrow p(0, y), q(y, z), z > th \end{aligned}$$

In the clause defining rh , let th be a threshold value that we assume to be always greater or equal 0. Now, let the risks be denied by the following integrity theory:

$$IC = \{\leftarrow rl(x), \leftarrow rm(x), \leftarrow rh(x)\}.$$

Before populating D with facts about p and q , let us verify that IC is satisfiable at all by any extension of D . Indeed, it is, e.g., by each extension of p such that no fact of the form $p(0, y)$ is in p and any of the following alternatives holds: either $p = q$, or D contains $\{q(2, 1), p(1, 2), p(2, 1)\}$ and arbitrarily many facts of the form $p(n, n + m)$, for $n > 1, m > 0$.

Now, let the extensions of p and q be as follows.

$$\begin{aligned} &p(0, 0), p(0, 1), p(0, 2), p(0, 3), \dots, p(0, 10000), \\ &p(1, 2), p(2, 4), p(3, 6), p(4, 8), \dots, p(5000, 10000) \\ &q(0, 0), q(1, 0), q(3, 0), q(5, 0), q(7, 0), \dots, q(9999, 0) \end{aligned}$$

Clearly, there is a single violated low-risk case in D , which is caused by $p(0, 0)$. Let us make sure that there is no other violated risk case in D , but trying to refute each denial about rl , rm and rh .

First of all, there obviously is no other low-risk cause of form $p(x, x)$ that would violate $\leftarrow rl(x)$.

Next, let us try to find an instance of the body of the first clause of rm that would be *true* in D . Since the second column of q is always 0, $q(x, 0), \sim p(0, x)$, would have to be *true*. That, however, cannot be, since $p(0, x) \notin D$ for each x such that $q(x, 0) \in D$.

For trying to find a satisfied instance of the body of the second clause of rm , let e stand for an even number greater or equal 0, o for an odd number greater or equal 1, and

n for any natural number greater or equal 0. Further note that each p -fact in D is either of the form $p(0, e)$ or $p(0, o)$ or $p(n, 2n)$, for $n > 1$. So, since the second column of p joins with the first column of q only if their value is an even number, the only possible instances of that clause which could make its body *true* are of one of the following three forms:

$$\begin{aligned}
 & p(0, e), q(e, z), \sim p(e, z), \sim q(z, 0) \\
 \text{or} & p(0, o), q(o, 0), \sim p(o, 0), \sim q(0, 0) \\
 \text{or} & p(n, 2n), q(2n, 0), \sim p(2n, 0), \sim q(0, n)
 \end{aligned}$$

Obviously, none of these instances can become *true*, because $q(e, z)$ does not hold for any z , $q(0, 0)$ is *true* in D , and $q(2n, 0)$ is *false* for each $n > 0$.

Last, the clause of rh : to make its body *true* would require that $0 > th$, but we have excluded that. Hence, we have verified that $\leftarrow rl(0)$ is the only violated risk case of IC in D , and that $p(0, 0)$ is its only cause.

Now, consider the update $U = \text{insert } q(0, 9999)$, for illustrating 1) - 6) above.

4.1 Brute-Force Risk Management

The cost of brute-force checking for any update is high. That is a commonplace, but let us see in some more detail to what brute-force evaluation of IC amounts, for later comparison.

Evaluation of $\leftarrow rl(x)$ involves a scan of all of p . Evaluation of $\leftarrow rm(x)$ involves joins of p and q , a join of local p with remote q , plus possibly many lookups in p and q . Evaluation of $\leftarrow rh(x)$ involves a join of local p with remote q , plus the evaluation of possibly many ground expressions of the form $z > th$.

With large extensions of p and q , the evaluation outlined above may last too long, particularly if safety-critical risks are monitored in real time. In 4.2, we shall see that it is far less expensive to use an inconsistency-tolerant method that simplifies the evaluation of integrity constraints by taking the update into account and by limiting its focus on the data that are affected by the update.

4.2 Inconsistency-Tolerant Risk Management

We are going to see that the cost of inconsistency-tolerant integrity checking of U is much lower than to use brute-force evaluation. But, before we go into details, recall that the use of any traditional method that insists on the satisfaction of IC in the old state D is prohibited for the database in our example, since $D(IC) = \text{false}$.

Typical simplification methods compile pre-simplifications for update patterns at constraint specification time. Thus, the cost of such pre-simplifications at update time is nil. U matches the update pattern $q(a, b)$, which in turn matches precisely the following unfoldings of $\leftarrow rm$ by the two clauses defining rm , and of $\leftarrow rh$, respectively.

$$\begin{aligned}
 & \leftarrow q(x, y), \sim p(y, x) \\
 & \leftarrow p(x, y), q(y, z), \sim p(y, z), \sim q(z, x) \\
 & \leftarrow p(0, y), q(y, z), z > th
 \end{aligned}$$

Thus, the pre-simplifications compiled for the patter for insertions of facts of the form $q(a, b)$, are as follows.

$$\begin{aligned} &\leftarrow \sim p(b, a) \\ &\leftarrow p(x, a), \sim p(a, b), \sim q(b, x) \\ &\leftarrow p(0, a), b > th \end{aligned}$$

Substituting (a, b) by the inserted values $(0, 9999)$ at update time yields the following simplifications.

$$\begin{aligned} &\leftarrow \sim p(9999, 0) \\ &\leftarrow p(x, 0), \sim p(0, 9999), \sim q(9999, x) \\ &\leftarrow p(0, 0), 9999 > th \end{aligned}$$

By a simple lookup of $p(9999, 0)$ for evaluating the first of the three denials, it is inferred that $\leftarrow rm$ is violated.

Since a medium risk has been detected, there is in principle no need to continue checking the remaining two simplified denials. However, we are going to do that, in order to build a bridge to point 3).

Evaluating the second denial from left to right amounts to the cost of answering the query $\leftarrow p(x, 0)$. The single answer is $x = 0$. Then, a lookup of $q(9999, 0)$ succeeds. Hence, the second denial is *true*, which means that there is no further medium risk.

Since $p(0, 0)$ is *true*, the third denial turns out to be violated if $9999 > th$ holds, which indicates a high security risk.

To conclude, let us summarize this subsection. Inconsistency-tolerant integrity checking of U essentially costs a simple access to the p relation. Only one more lookup is needed if all constraints are evaluated. And, perhaps more importantly, inconsistency-tolerant integrity checking prevents medium- and high-risk violations that would be caused by the update if it were not rejected.

4.3 Dependable Risk Management

Inconsistency-tolerant constraint checking is dependable. It is not if a non-inconsistency-tolerant method (e.g., as described in [9,10]) is used. This claim is substantiated by the inconsistency-intolerant reasoning as outlined below.

Since the p relation is not affected by U , the truth value of the unfolding $\leftarrow p(x, x)$ of the constraint $\leftarrow rl(x)$ is the same in D and D^U . Since each method that is not inconsistency-tolerant insists on the premise that all constraints be satisfied in the old state, such methods, when applied to our example, conclude that the unfolded denial $\leftarrow p(x, x)$ is *true* in D and D^U , even though $p(0, 0) \in D$. That conclusion is then applied to the third of the simplified unfoldings from 2), i.e., to $\leftarrow p(0, 0), 9999 > th$, which thus is wrongly concluded to be satisfied in D^U if $9999 > th$ holds.

The subsumption-based reasoning of methods that are not inconsistency-tolerant can be summarized as follows: Applying the premise that $\leftarrow p(x, x)$ is satisfied to $\leftarrow p(0, 0), 9999 > th$ infers that the latter also remains satisfied in D^U , because it is subsumed by $\leftarrow p(x, x)$. Thus, non-inconsistency-tolerant integrity checking may wrongly conclude that the high risk constraint $\leftarrow rh(z)$ is not violated in D^U .

4.4 Repair-Based Risk Management (Repairing the Old State)

The traditional integrity checking approach insists on total constraint satisfaction in the old state. To comply with that, in order to use traditional approaches for risk management, all extant violations need to be repaired before each update. Yet, we are going to see that repairing the old state is costly.

In general, the identification of all extant violations may already be very expensive in large databases, and indeed unaffordable at update time. Fortunately, however, there is only a single low-risk constraint violation in our example, as we have already seen before: $p(0, 0)$ is the only cause of the only constraint violation $\leftarrow rl(0)$ in D . Thus, to repair D means to request the update $R = delete\ p(0, 0)$, and to execute R if it preserves all constraints.

To check R for integrity preservation means to check the simplified denials

$$\leftarrow q(0, 0)$$

and

$$\leftarrow p(x, 0), q(0, 0), \sim q(0, x)$$

obtained from resolving $\sim p(0, 0)$ with the bodies of the two clauses defining rm , since precisely those two clauses are affected by the deletion of $p(0, 0)$. Hence, no constraint other than $\leftarrow rm(y)$ is potentially violated by the intended repair.

Of the two simplified denials above, the second one clearly is satisfied in D^R , since no fact of the form $p(x, 0)$ remains in the database after $p(0, 0)$ is deleted. However, the first one is violated, since $q(0, 0)$ is *true* in D^U . Hence, another repair action is needed. The obvious candidate is *delete* $q(0, 0)$. That update request affects the constraint

$$rm(y) \leftarrow p(x, y), q(y, z), \sim p(y, z), \sim q(z, x)$$

and yields the simplified check of

$$\leftarrow p(0, y), q(y, 0), \sim p(y, 0).$$

Obviously, this denial is violated by facts in D that are of the form $p(0, o)$ and $q(o, 0)$, where o is an odd number in the interval $[1, 9999]$. Thus, to delete $q(0, 0)$ for repairing the violation caused by deleting $p(0, 0)$ causes the violation of each case of the form $\leftarrow rm(o)$, for each odd number o in $[1, 9999]$.

Clearly, many facts about p or q would have to be deleted in order to repair each of these violated cases. For simplicity, we won't follow them through, since the point that repairing D is very complex and tends to be much more expensive than inconsistency-tolerant integrity checking has become obvious already. We only recall the big advantage of inconsistency-tolerant integrity checking that repair actions do not have to take place at update time. Instead, they can be taken off-line, at any convenient moment.

4.5 Repair-Based Risk Management (Repairing the New State)

Also repairing the new state is more costly than to simply tolerate extant constraint violations until they can be repaired at some better moment. In our example, this becomes obvious by recalling from 1) that, in D^U , there are three violated cases: the low-risk case that is already violated in D and the medium- and high risk cases as detected by inconsistency-tolerant integrity checking. To repair them is indeed even more complicated than to only repair the violated low-risk case, as attempted in 4).

Moreover, it should be noted for risk management that it is no good idea in general to simply accept an update without checking for potential violations of constraints, and to attempt repairs only after the update is committed, because repairing takes time, during which an updated but unchecked state may contain malicious risks of any order.

4.6 Consistent Query Answering for Risk Management?

Consistent query answering in inconsistent databases (CQA) is a popular approach to cope with extant constraint violations for query answering [1]. Although query answering is not the topic of this paper, a connection between inconsistency-tolerant integrity checking and CQA can easily be drawn, because the monitoring of risk constraints involves their evaluation. Thus, the idea may arise to use CQA for evaluating constraints as queries, in order to avoid wrong answers that could be due to extant constraint violations.

Unfortunately, to evaluate constraints or simplifications thereof by CQA is not recommendable, because consistent answers are defined to be those that are true in each minimally repaired state of the database. Thus, for each queried constraint, CQA will by definition return the empty answer, which indicates the satisfaction of the constraint. Thus, answers to queried constraints that are computed by CQA have in fact no meaningful interpretation.

For instance, CAQ computes the empty answer to the query $\leftarrow rl(x)$ as well as to the query $\leftarrow rh(z)$, for any extension of the relations p and q . However, the only sensibly correct answer to the first query in D is $x = 0$. Similarly, the only reasonable answer to the second query in D^U is $x = 9999$, assuming that $9999 > th$. These answers are reasonable because they correctly indicate risks contained in D and D^U , respectively.

This shows that, despite of many unquestionable merits of CQA, it should not be used for monitoring risks if risks are modeled by integrity constraints.

5 Related Work

Although semantic properties such as integrity and risks associated to data are intuitively related, they never have been approached in a uniform manner, to the best of our knowledge, neither in theory nor in practice.

Kinships and semantic differences between the integrity of data and data that involve some sort of risk, in the sense of being uncertain, are observed in a collection of work on modeling and managing uncertain data [11]. In that book, largely diverse approaches to handle data that involve some risk of uncertainty are proposed. In particular, approaches such as probabilistic and fuzzy set modeling, exception handling, repairing and paraconsistent reasoning are discussed. However, no particular approach to integrity checking is considered.

Similar to uncertainty, also quality impairment can be understood as a special kind of riskiness of data. Several paraconsistent logics that tolerate inconsistency and quality impairment of data have been proposed, e.g., in [7,2]. Each of them, however, departs from classical first-order logic, by adopting some annotated, probabilistic, modal

or multivalued logic, or by replacing standard axioms and inference rules with non-standard axiomatizations. As opposed to that, inconsistency-tolerant integrity checking fully conforms with standard datalog and does not need any extension of classical logic.

Further work on the management of semantic inconsistencies in databases is going on in the field of measuring inconsistency [8,5]. As shown in [5], inconsistency measures can be used for a form of inconsistency-tolerant integrity checking that is different from the approach outlined in section 3. It accepts an update only if the measured amount of inconsistency in the old state does not increase in the new state. There are several possible ways to measure inconsistency. Two that are directly related to section 3 are to count the number of violated cases, or to compare the set of violated cases before and after the update [4]. In [5], some more measures related to inconsistency-tolerant integrity checking are discussed.

6 Conclusions

We have shown that risks can be modeled by integrity constraints and monitored by inconsistency-tolerant integrity checking methods.

Traditionally, no method for simplified integrity checking has tolerated a database that is inconsistent with its constraints. However, as shown in [6], it is possible to waive that intolerance without any penalty. Hence, assertions that model risk constraints, the occasional violation of which is tolerable, can be monitored efficiently by inconsistency-tolerant integrity checking methods.

As illustrated in the extended example of section 4, inconsistency tolerance is essential, since wrong, possibly fatal conclusions can be inferred from deficient data by using a method that is not inconsistency-tolerant. Many methods that are inconsistency-tolerant, and some that are not, have been identified in [4,5,6].

Ongoing work is concerned with establishing a closer relationship of inconsistency-tolerant integrity checking with the fields of repairing, inconsistency measuring and consistent query answering.

References

1. Arenas, M., Bertossi, L., Chomicki, J.: Consistent query answers in inconsistent databases. In: 18th PODS, pp. 68–79. ACM Press, New York (1999)
2. Bertossi, L., Hunter, A., Schaub, T. (eds.): Inconsistency Tolerance. LNCS, vol. 3300. Springer, Heidelberg (2005)
3. Bowen, K., Kowalski, R.A.: Amalgamating language and metalanguage. In: Clark, K., Tärnlund, S.A. (eds.) Logic Programming, pp. 153–172. Academic Press, London (1982)
4. Decker, H., Martinenghi, D.: Classifying integrity checking methods with regard to inconsistency tolerance. In: 10th PPDP, pp. 195–204. ACM Press, New York (2008)
5. Decker, H., Martinenghi, D.: Modeling, Measuring and Monitoring the Quality of Information. In: Heuser, C.A., Pernul, G. (eds.) ER 2009. LNCS, vol. 5833, pp. 212–221. Springer, Heidelberg (2009)
6. Decker, H., Martinenghi, D.: Inconsistency-tolerant Integrity Checking. IEEE Transactions on Knowledge and Data Engineering. Abstract and preprint available at (2010), <http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.87> (to appear)

7. Decker, H., Villadsen, J., Waragai, T. (eds.): ICLP 2002 workshop on Paraconsistent Computational Logic. Datalogiske Skrifter, vol. 95. Roskilde University, Denmark (2002)
8. Grant, J., Hunter, A.: Measuring inconsistency in knowledgebases. *J. Intelligent Information Systems* 27(2), 159–184 (2006)
9. Gupta, A., Sagiv, Y., Ullman, J., Widom, J.: Constraint checking with partial information. In: 13th PODS, pp. 45–55. ACM Press, New York (1994)
10. Lee, S.Y., Ling, T.W.: Further improvements on integrity constraint checking for stratifiable deductive databases. In: 22nd VLDB, pp. 495–505. Morgan Kaufmann, San Francisco (1996)
11. Motro, A., Smets, P.: *Uncertainty Management in Information Systems: From Needs to Solutions*. Kluwer, Dordrecht (1996)
12. Ramakrishnan, R., Gehrke, J.: *Database Management Systems*. McGraw-Hill, New York (2003)

Energy Efficient Data Sorting Using Standard Sorting Algorithms

Christian Bunse¹, Hagen Höpfner², Suman Roychoudhury³, and Essam Mansour⁴

¹ University of Applied Sciences Stralsund

Zur Schwedenschanze 15, 18439 Stralsund, Germany

² Bauhaus University of Weimar, Bauhausstr. 11, 99423 Weimar, Germany

³ Tata Research Development and Design Center

54-B, Hadapsar Industrial Estate, Pune, MH, 411013, India

⁴ King Abdullah University of Science and Technology

Thuwal 23955-6900, Kingdom of Saudi Arabia

Christian.Bunse@fh-stralsund.de,

<http://www.imenco.eu>,

hoepfner@acm.org,

<http://www.hoepfner.ws>,

suman.roychoudhury@tcs.com,

essam.mansour@ieee.org,

<http://elab.ws>

Abstract. Protecting the environment by saving energy and thus reducing carbon dioxide emissions is one of today's hottest and most challenging topics. Although the perspective for reducing energy consumption, from ecological and business perspectives is clear, from a technological point of view, the realization especially for mobile systems still falls behind expectations. Novel strategies that allow (software) systems to dynamically adapt themselves at runtime can be effectively used to reduce energy consumption. This paper presents a case study that examines the impact of using an energy management component that dynamically selects and applies the "optimal" sorting algorithm, from an energy perspective, during multi-party mobile communication. Interestingly, the results indicate that algorithmic performance is not key and that dynamically switching algorithms at runtime does have a significant impact on energy consumption.

Keywords: Energy awareness, Software engineering, Adaptivity, Mobile information systems.

1 Introduction

Mobile and embedded devices become more and more important in all areas of the daily life. Nowadays, they also form the basis of the ubiquitous or pervasive computing paradigm. Information is accessible everywhere at any time via mobile phones, small sensors collectively measure environmental parameters, and micro controller based computers are embedded in many devices starting from toys and ending with cars or air planes. In fact, the landscape of embedded computing changes dramatically. Unfortunately, all these small components share a need for a (mobile) power supply. As

smaller the device, as more the uptime depends on the efficient usage of the limited energy resource. So, all parts of a mobile or embedded system have to be energy aware.

Recently significant research effort has been spent on optimizing hardware related energy consumption [1]. However, less importance has been given towards energy efficient usage of hardware components by optimizing the underlying software. This particular aspect of software optimization is addressed in this paper. Devices contain several hardware components (e.g. CPU, external memory, communication devices), which have different levels of energy consumption (cf. [13]). Therefore, the software must be able to adapt itself to meet the underlying user requirements while conserving maximum energy. In previous works [11] we therefore introduced the concept of resource substitution. Preliminary results [4] have shown that different implementations of algorithms result in varying energy consumptions. In particular, we implemented various sorting algorithms because sorting efficiency is relevant to almost all applications. Furthermore, many database management algorithms that implement join (e.g. Sort-Merge-Join) or set operations implicitly use sorting algorithms. Our experiments revealed that memory intensive implementations consumed much more energy than in-place implementations i.e., Quicksort in general consumed more energy than Insertion-sort. However, if processing speed is given priority over energy, Insertionsort performs slower than Quicksort, thus resulting in a longer execution time. Therefore the question is, how much data can be sorted by an algorithm implementation by keeping an optimal balance between energy consumption and processing speed.

In this paper we present our approach for energy saving software by choosing the appropriate algorithm. Based on experimental results, we introduce trend functions for each implementation of the examined sorting algorithms. These trend functions are then used to decide on which algorithm to use under certain conditions or based on the users needs (faster speed vs. saving energy). Furthermore, we describe a dynamic optimization approach that changes the used implementation at runtime.

Please note, this paper is an extended version of [5]. Based on the reviews and the conference discussions organizers selected the paper for the publication in this book.

The remainder of this paper is structured as follows: Section 2 describes the related work. Section 3 briefly introduces the researched sorting algorithms. Section 4 introduces the optimizer component and the trend functions. Section 5 explains the experimental setup and the experiments performed as proof of concept. Section 6 includes the interpretation of the experimental results. Section 7 discusses the validity of these results. Section 8 summarizes the paper and gives an outlook to future research.

2 Related Work

Due to the orientation towards mobile- and embedded systems, several research projects have been conducted regarding the topic of energy consumption. Optimizing energy consumption is one of the most fundamental factors for an efficient battery-powered system. Research on energy consumption falls into one of the following categories: (1) Hardware, or (2) Software [14]. Research that belongs to the hardware category, attempts to optimize the energy consumption by investigating hardware usage, such as [6,17], and innovating new hardware devices and techniques, such as [25,26]. Research in the second category attempts to understand how the different methods and

techniques of software affect energy consumption. Research in this category can be further classified according to the main factors affecting energy consumption: *networking*, *communication*, *application nature*, *memory management*, and *algorithms*. Concerning *networking* work such as [7,21], provide new routing techniques that are aware of energy consumption. Other efforts of this category focus on providing energy-aware protocols for transmitting data in wireless networks [20,22] and ad-hoc networks [9]. Memory consumption is an important factor concerning a system's energy consumption. In this regard work such as [15,18] have provided energy-aware memory management techniques. In battery-powered systems, it is not sufficient to analyze algorithms based only on time and space complexity. Energy-aware algorithms such as [14] supporting randomness, [19] focusing on cryptographic, and [24] investigating into wireless sensor networks were published.

3 Sorting Algorithms

In the first days of computing, sorting data (numbers, names, etc.) was in focus of research. One reason might be that although sorting appears to be “easy”, its efficient execution by machines is inherently complex. Even today, sorting algorithms are still being optimized or even newly invented. When it comes to mobile systems and information retrieval, efficient sorting is a major concern regarding performance and energy consumption. In the following we describe the set of sorting algorithms that were used in the context of this study. This set was defined to include major algorithms that are either used in form of library functions (e.g., Quicksort), are easily programmable (e.g., Bubblesort) or that are regularly taught to computer science students. In other words, our goal was to cover those algorithms that are in widespread use. More details on them can be found in standard text books on algorithms and data structures such as [16].

- **Bubblesort** belongs to the family of comparison-based sorting. It works by repeatedly iterating through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order. The worst-case complexity is $O(n^2)$ ¹ and the best case is $O(n)$. Its memory complexity is $O(n)$.
- **Heapsort** is a comparison-based sorting algorithm and part of the selection sort family. Although somewhat slower in practice on most machines than a good implementation of Quicksort, it has the advantage of a worst-case time complexity of $O(n \log n)$.
- **Insertionsort** is a naive algorithm that belongs to the family of comparison-based sorting. In general insertion sort has a time complexity of $O(n^2)$ but is known to be efficient on data sets which are already substantially sorted. Its average complexity is $O(n^2/4)$ and linear ($O(n)$) in the best case. Furthermore insertion sort is an in-place algorithm that requires a linear amount $O(n)$ of memory space.
- **Mergesort** was invented by John von Neumann and belongs to the family of comparison-based sorting. Mergesort has an average and worst-case performance of $O(n \log n)$. Unfortunately, Mergesort requires three times the memory of in-place algorithms such as Insertionsort.

¹ n represents the size of input; the number of elements to be sorted.

- **Quicksort** was developed by Sir Charles Antony Richard Hoare [10]. It belongs to the family of exchange sorting. On average, Quicksort makes $O(n \log n)$ comparisons to sort n items, but in its worst case it requires $O(n^2)$ comparisons. Typically, Quicksort is regarded as one of the most efficient algorithms and is therefore typically used for all sorting tasks. Quicksort’s memory usage depends on factors such as choosing the right Pivot-element, etc. On average, having a recursion depth of $O(\log n)$, the memory complexity of Quicksort is $O(\log n)$ as well.
- **Selectionsort** belongs to the family of in-place comparison-based sorting. It typically searches for the minimum value, exchanges it with the value in the first position and repeats the first two steps for the remaining list. On average Selectionsort has a $O(n^2)$ complexity that makes it inefficient on large lists. Selectionsort typically outperforms bubble sort but is generally outperformed by Insertionsort.
- **Shakersort** [2] is a variant of Shellsort that compares each adjacent pair of items in a list in turn, swapping them if necessary, and alternately passes through the list from the beginning to the end then from the end to the beginning. It stops when a pass does no swaps. Its complexity is $O(n^2)$ for arbitrary data, but approaches $O(n)$ if the list is nearly in order at the beginning.
- **Shellsort** is a generalization of Insertionsort, named after its inventor, Donald Shell. It belongs to the family of in-place sorting but is regarded to be unstable. It performs $O(n^2)$ comparisons and exchanges in the worst case, but can be improved to $O(n \log_2 n)$. This is worse than the optimal comparison-based sorts, which are $O(n \log n)$. Shellsort improves Insertionsort by comparing elements separated by a gap of several positions. This lets an element take “bigger steps” toward its expected position. Multiple passes over the data are taken with smaller and smaller gap sizes. The last step of Shellsort is a plain Insertionsort, but by then, the list of data is guaranteed to be almost sorted.

4 Optimizing Energy

To dynamically adapt/optimize the energy consumption of a mobile and/or embedded system (i.e., the sorting node used throughout our experiment), we developed an architecture (see Figure 1) that closely follows the idea of the energy management component presented in [3]. EMC aimed at optimizing the communication effort of a system. Here the focus is on using the “optimal” algorithm concerning energy and processing speed.

At its core, the experiment system defines a family of sorting strategies (algorithms), encapsulates each of these, and makes them interchangeable. This is nicely supported by the strategy pattern defined by [8]. The strategy pattern supports the development of software systems as a loosely coupled collection of inter-changeable parts. The pattern decouples a strategy from its host, and encapsulates it into a separate class. It thus, supports the separation of an object and its behaviour by organizing them into two different classes. This allows switching to the strategy that is needed at a specific time.

There are several advantages of applying the strategy pattern for an adaptable software system. First, since the system has to choose the most appropriate strategy concerning performance and energy it is simpler to keep track of them by implementing

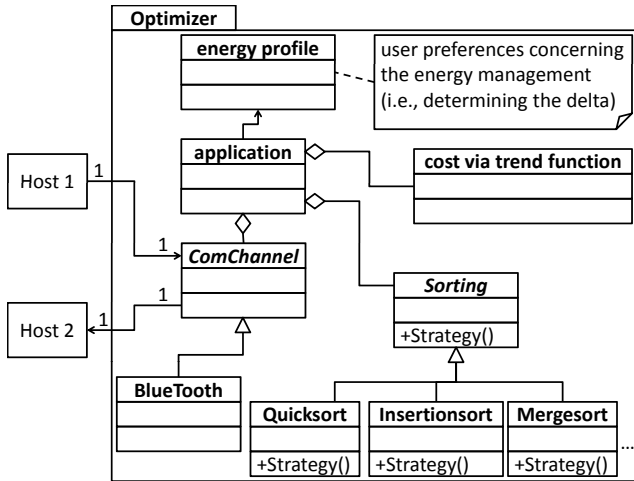


Fig. 1. Algorithm-Energy Optimizer

each strategy by a separate class instead of embedding it in the body of a method. Having separate classes allows simply adding, removing, or changing any of the strategies. Second, the use of the strategy pattern also provides an alternative to sub-classing an object. This also avoids the static behavior of sub-classing. Changes therefore require the creation/instantiation of a different subclass and replacing that object with it. The strategy pattern allows switching the object's strategy, and it will immediately change how it behaves. Third, using the strategy pattern also eliminates the need for various conditional statements. When different strategies are implemented within one class, it is difficult to choose among them without resorting according to the conditional statements. The strategy pattern improves this situation since strategies are encapsulated as an object that is interchangeable at runtime.

To select the most appropriate strategy/algorithm, the optimizer has to be aware of the cost of its execution with respect to energy. Therefore, a cost model is needed that provides a "rough" estimate of an algorithm's energy consumption based on the input size. However, it has to be noted that the used cost model instance is only valid for the actually used platform. Basically we followed the empirical data gathered in the context of [4] concerning the energy consumption of sorting algorithms running on AVR processors.

We used the extrapolated trend functions of the different sorting algorithms (see Figure 2) as a basis for the cost functions. The trend function calculates an estimation of the required energy for 1,000 executions² of an algorithm, based on the input size n . In detail, the trend functions listed in Table 1 were extrapolated, whereby the R^2 value that represents the goodness of fit was 1.

Since our goal was to find the optimal balance between sorting performance and energy consumption it was not sufficient to simply use the trend functions as cost function. Due to the linear nature of the trend function the result would always indicate

² To level out measurement errors.

Table 1. Trend functions

Algorithm	trend function
Quicksort	$F(n) = n^{1.1388} \cdot 0.1533$
RQuicksort	$F(n) = n^{1.1771} \cdot 0.1467$
Insertionsort	$F(n) = n^{1.0679} \cdot 0.09121467$
Mergesort	$F(n) = n^{1.1035} \cdot 0.1912$
RMergesort	$F(n) = n^{1.1384} \cdot 0.3221$
Heapsort	$F(n) = n \cdot 0.2324 + 0.0286$
Shellsort	$F(n) = n^2 \cdot 0.0071 + n \cdot 0.0047$ $+0.0939$
Selectionsort	$F(n) = n^2 \cdot 0.013 - n \cdot 0.0236$ $+0.1908$

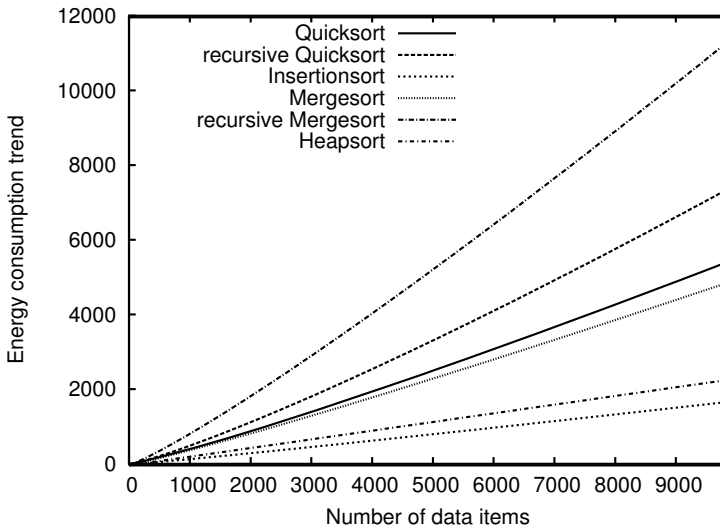


Fig. 2. Trend functions - excerpt

Insertionsort as the most energy-efficient algorithm. Keeping these formulas and assumptions in mind the following selection algorithm can be defined:

1. By using the size n of the set as an input the energy-related costs for all algorithms are calculated and stored.
2. The minimum result and thus the most energy-efficient algorithm is identified.
3. Based on the algorithmic complexity, the minimum value is compared to those values that are related to algorithms of “lower” complexity classes.
4. If the difference in energy-consumption is below a predefined threshold or delta the “faster” algorithm is chosen.

The goal of an adaptive application (e.g., our experimental system) is to optimize the Quality-of-Service (QoS) perceived by the user. Unfortunately, often optimization approaches either enforce predetermined (fixed) policies or offer only limited mechanisms

for controlling optimization. According to [23], these limitations prevent adaptive systems from addressing these important issues. User goals often entail trade-offs among different aspects of quality (e.g., enhancing battery life or faster execution times).

The Optimizer architecture (see Figure 1) allows users to actually determine the trade-off between performance and energy consumption, simply by changing the cost function delta. As larger the delta becomes as higher the sorting performance and as lower the battery-lifetime of the system. Thus, the delta determines the size of data-sets to be sorted by a specific algorithm. In the context of our experiments we experimented with different delta values and observed that defining the delta as 1, 200 provides the “best” optimization results.

5 Experimental Design

Our previous experiments provided some insight into the area of software-related energy consumption. In these experiments we collected data concerning the energy consumption of sorting algorithms as well as algorithms that apply them. The results show that energy consumption is driven by factors such as memory consumption and performance leading to the fact that the fastest algorithm (e.g., Quicksort) is not the most efficient algorithm concerning energy-consumption.

In this sense we developed a simple system (see Figure 3). An embedded node wirelessly receives data-sets, sorts them and sends the sorted set to another recipient. The goal of the node is to primarily sort data. However, since the node is battery powered and the end-user expects short response times, sorting is optimized according to response time and average energy consumption (i.e., maximize up-time). On application scenario for such a node is a wireless sensor network that collects position or life-data of cattle. Nodes are communicating wirelessly (e.g., ZigBee or Bluetooth), whereby the controller-node (i.e., the node that pre-processes the data) finally provides the pre-processed data to a PC. In the context of our experiments we use a simplified version for brevity of illustration.

The system comprises a micro controller (i.e., ATmega128, external SRAM, running on a STK500/501 board) and two Bluetooth interfaces (i.e., BlueSmirf modules). The system establishes two data-connections to different hosts via its Bluetooth modules. It reads values (variable size sets of unsorted integers) via one data-connection, selects the most appropriate sorting algorithm according to its optimization goal, and transmits the sorted set to another host via the second data-connection.

5.1 Experimental Runs

Within this experiment we compared three different approaches to optimization concerning their performance (time and size) and their energy consumption. In the first run the data was sorted by only using the Quicksort algorithm. Thus, this run was optimized for speed. In the second run, the data was sorted by only using the Insertionsort algorithm. This represents an optimization for energy-efficiency (max. battery lifetime) based on our previously reported findings[4]. In the third run we then made use of the algorithm optimizer/selector (see Figure 1). The Optimizer aims at balancing speed

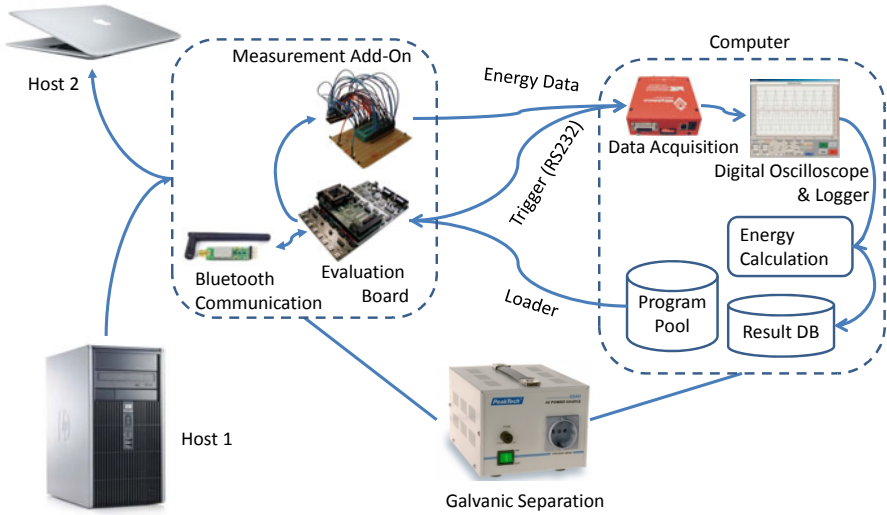


Fig. 3. Experiment System Overview

(performance) and energy efficiency to increase the amount of sorted data and processing speed while at the same time increasing battery lifetime through energy savings.

5.2 Data Collection

As described previously the experiment system is connected to two different hosts via Bluetooth (i.e., Figure 3 provides an overview). It receives sets (i.e., sequences of random integer values) of varying sizes from the first host. The actual size of a transmitted set is limited to 1,000 elements but is randomly chosen.

According to the actual experimental run, the system either applies a specific sorting algorithm (i.e., Quicksort or Insertionsort) or selects the “best” sorting algorithm from a set of algorithms and applies the selected algorithm to the received set. The sorted set is then sent to Host 2 and the next data set is received. This cycle is executed until the battery is empty.

Send and receive (Host 1 and Host 2) are synchronized by the clock. This leaves sufficient time for sorting and retransmission. In addition the Bluetooth modules (externally powered) provide send/receive buffers to level out overlaps.

During the experimental runs the following data was collected:

- The size of every set to be sorted (i.e., n). The size was randomly chosen but limited to a maximum of 1,000 elements.
- The number of sets sorted (i.e., N). This is the overall number of successfully executed sorting requests throughout the experimental run.
- The battery level (i.e., V). V was measured at fixed time intervals, whereby we assume that the actual voltage level of a battery indicates its status and charge level.

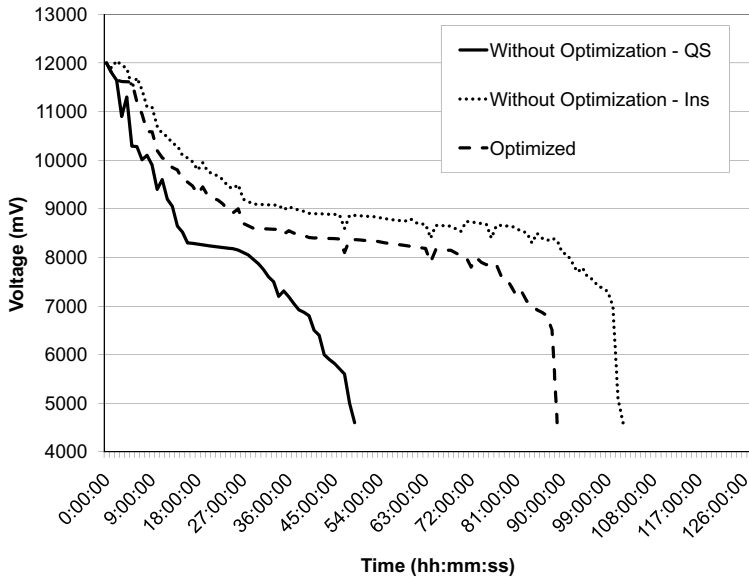


Fig. 4. Battery Lifetime

- Run and Execution Time (i.e., P). Run and execution time was measured by hosts one and two (i.e., time when a set was sent and when the sorted set was received). Times were not measured at the target platform in order to not falsify the measurements.

6 Evaluation of Results

Initial measurement within the experimental runs shows that optimization results are as expected. When looking at the battery level V over time (i.e., up-time of our system) it supports the initial assumption that the uptime of a systems is directly related towards the energy consumption related to the executed software system. However, a closer look at Figure 4 shows that a non-adaptive approach either results in an excellent or a poor energy efficiency. Interestingly, the results for the adaptive are close to those of the non-optimized Insertionsort variant.

When recalling the results of Figure 4 the question arises why should we adapt the system or is optimization really necessary. It seems that by choosing a specific algorithm better results can be achieved than by dynamic adaption. Therefore, we have to examine the performance of the different variants concerning the amount of sorting requests and the amount of data.

Figure 5 supports the initial assumption concerning the trade-off between energy efficiency and performance. High-performing variants (i.e., Quicksort) handle more sorting requests (i.e., N) in a shorter period of time but result in a very limited V . Energy-efficient variants (i.e., Insertionsort) result in an optimal V but handle significantly less sorting requests. Only adaptive (i.e., optimized) systems provide a good balance of energy-efficiency and performance.

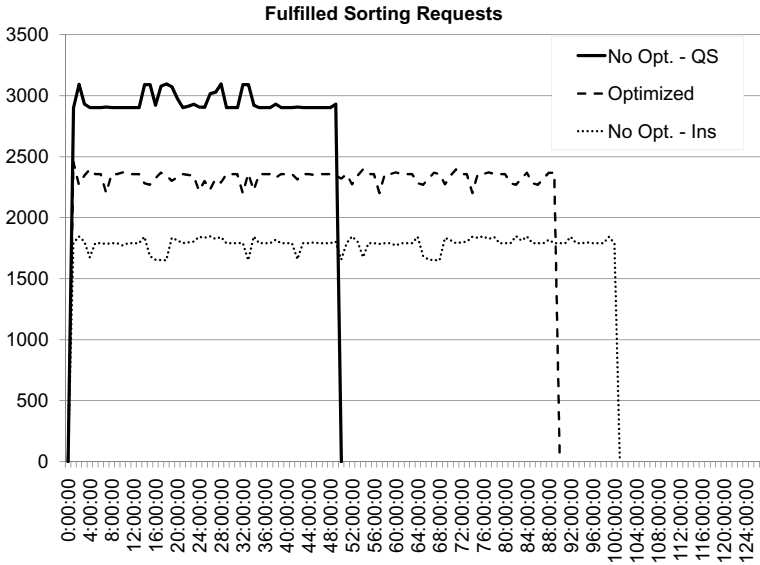


Fig. 5. Request Performance

This is also supported by Figure 6, which shows the total number of elements that were sorted over time. Measurement results expose a linear growth that roughly represent the sums of sorting requests sizes. The results confirm our initial assumption that an optimization for speed (Quicksort variant) results in a fast growing curve that covers a short time period. Optimization for energy (Insertionsort variant) results in a curve that spans a broad time range but that does not grow as fast as the Quicksort related curve. Finally, the optimized system seems to combine the advantages of other approaches. It covers a broad time range and sorts a high number of elements. In others words, the optimized system variant provides a well-balanced behavior regarding performance and energy consumption.

The findings concerning the effects of dynamically choosing an algorithm at runtime are also supported by the fact that this approach sorts more data in total than the other two approaches. Thus, optimization does not provide results somewhere between those of the non-adaptive systems but uses their synergy effects to provide even better results.

7 Threats to Validity

The authors view this study as exploratory. Thus, threats limit generalization of this research, but do not prevent the results from being used in further studies.

Construct Validity is the degree to which the independent and dependent variables accurately measure the concepts they purport to measure. In specific, energy consumption is a difficult concept to measure. In the context of this paper it is argued that the chosen approach (assessing the battery voltage level V) is an intuitively reasonable measure. Of

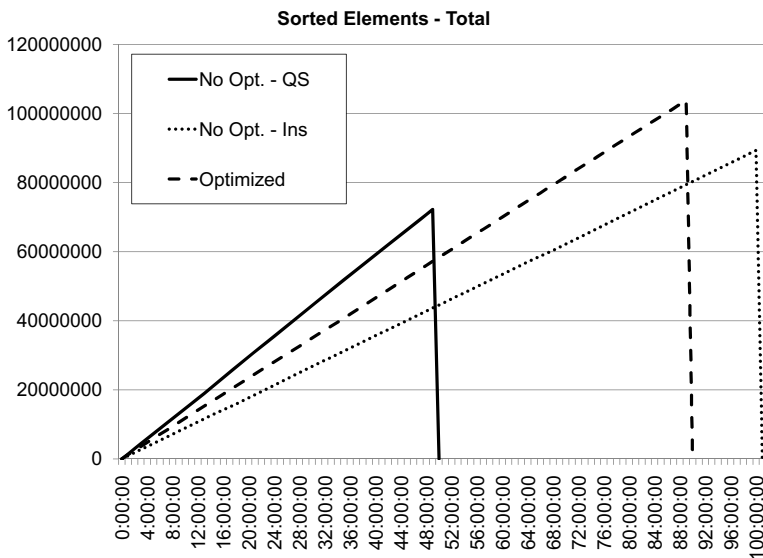


Fig. 6. Total Number of Sorted Elements

course, there are several other dimensions of the energy measurement problem but this is future work. However, in a simple study it is unlikely that all the different dimensions of a concept can be captured.

Internal Validity is the degree to which conclusions can be drawn about the causal effect of independent variables on the dependent variable. In specific, a history effect may result from measuring systems at different times (varying context temperature). Additional experiments and runs have shown that the temperature effect in a heated lab room can be neglected.

External Validity is the degree to which the results of the research can be generalized to the population under study and other research settings. In specific, the materials (platforms, software, etc.) may not be representative in terms of size and complexity. However, experiments in a university context do not allow the use of realistic systems for reasons such as cost, availability, etc. However, the authors view this study as exploratory and view its results as indicators that allow further research.

In order to improve the empirical study and address some of the threats to validity identified above, the following actions can be taken:

- Improve data collection. Data collection can be improved in several ways. First, by measuring not only battery voltage as an indirect energy consumption indicator but also the exact energy consumption, in joule, of the platform regarding each algorithmic run. Second, by increasing the sampling/measurement frequency (e.g., have sampling rates of 1 microsecond or below). A third option would be to automate the whole experimental procedure, thereby making time collection trivial.

- Improve the distinction of algorithmic complexities. The actual experiment used random data that was only comparable between runs. However, we did not make any distinction regarding best-, worst- and average-case data. By separating and explicitly distinguishing between these cases would allow for fine-grained analysis.
- Improve the generalizability of results by running the experiment on different platforms. Currently, results are limited to the AVR processor family and can thus, only serve as an indicator of the general situation. Therefore the experiment needs to be replicated on different platforms to get more and more reliable data.

8 Summary and Conclusions

Given the rising importance of mobile and small embedded devices, energy consumption becomes increasingly important. Current estimates by EUROSTATS predict that in 2020 10-35 percent (depending on which devices are taken into account) of the global energy consumption is consumed by computers and that this value will likely rise [3]. Therefore, means have to be found to reduce the energy consumption of such devices.

The focus of this paper is on dynamically adapting a simple system at runtime by algorithm substitution as a means for energy saving. Following the ideas of resource substitution strategies as presented in [11] we presented an Optimizer Component that follows the ideas of the dynamic energy management component EMC [3] and that can be plugged into other component based systems.

At its core the optimizer uses the energy-related trend functions of different sorting algorithms. In detail, the optimizer uses the different trend functions for determining the energy-related cost of a specific algorithm with respect to the algorithm's input-size. It then compares the results and uses a user-defined delta for selecting the best algorithm.

Our initial results are based on a micro-controller system (AVR processor family, cf. [12]) that communicate wirelessly by BlueTooth. The main system functionality is to receive data of varying size, sort it and to send it to another host. The data collected for different system variants was then used to examine if energy-consumption and sorting performance can be significantly improved. The collected data reveals that by optimization the amount of sorted data, battery lifetime. Moreover, the overall performance can be significantly increased. The experiments show the impact of software onto a systems energy consumption and a way to easily optimize a system in this regard.

To systematically evaluate the observed effects and to rule out the aforementioned threats to validity we currently prepare a case study for mobile information systems running on a PDA or Smartphone.

References

1. Bardine, A., Foglia, P., Gabrielli, G., Prete, C.A.: Analysis of static and dynamic energy consumption in NUCA caches: initial results. In: Proceedings of the 2007 workshop on MEMory performance: DEaling with Applications, systems and architecture, pp. 105–112. ACM, New York (2007)
2. Brejová, B.: Analyzing variants of Shellsort. *Information Processing Letters* 79(5), 223–227 (2001)

3. Bunse, C., Höpfner, H.: Resource substitution with components — Optimizing Energy Consumption. In: Cordeiro, J., Shishkov, B., Ranchordas, A.K., Helfert, M. (eds.) *Proceedings of the 3rd International Conference on Software and Data Technologie*, vol. SE/GSDCA/MUSE, pp. 28–35. INSTICC press, Setúbal (2008)
4. Bunse, C., Höpfner, H., Mansour, E., Roychoudhury, S.: Exploring the Energy Consumption of Data Sorting Algorithms in Embedded and Mobile Environments. In: *Proceedings of the 10th International Conference on Mobile Data Management: Systems, Services and Middleware*, Taipei, Taiwan, May 18–20, pp. 600–607. IEEE Computer Society Press, Los Alamitos (2009)
5. Bunse, C., Höpfner, H., Roychoudhury, S., Mansour, E.: Choosing the “best” Sorting Algorithm for Optimal Energy Consumption. In: *Proceedings of the 4th International Conference on Software and Data Technologie (ICSOFT 2009)*, Setúbal, Portugal, July 26–29, vol. 2, pp. 199–206. INSTICC press, Setúbal (2009)
6. Chen, J.-J., Thiele, L.: Expected system energy consumption minimization in leakage-aware DVS systems. In: *Proceeding of the Thirteenth International Symposium on Low Power Electronics and Design, ISLPED 2008*, pp. 315–320. ACM, New York (2008)
7. Feeney, L.M.: An Energy Consumption Model for Performance Analysis of Routing Protocols for Mobile Ad Hoc Networks. *Mobile Networks and Applications* 6(3), 239–249 (2001)
8. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Reading (1995)
9. Gurun, S., Nagpurkar, P., Zhao, B.Y.: Energy consumption and conservation in mobile peer-to-peer systems. In: *Proceedings of the 1st International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking, MobiShare 2006*, pp. 18–23. ACM, New York (2006)
10. Hoare, C.A.R.: Quicksort. *Computer Journal* 5(1), 10–15 (1962)
11. Höpfner, H., Bunse, C.: Resource Substitution for the Realization of Mobile Information Systems. In: Filipe, J., Helfert, M., Shishkov, B. (eds.) *Proceedings of the 2nd International Conference on Software and Data Technologie*, vol. Software Engineering, pp. 283–289. INSTICC Press, Setúbal (2007)
12. Höpfner, H., Bunse, C.: Energy Aware Data Management on AVR Micro Controller Based Systems. *ACM SIGSOFT Software Engineering Notes* 35(3) (May 2010)
13. Höpfner, H., Bunse, C.: Towards an energy-consumption based complexity classification for resource substitution strategies. In: Balke, W.T., Lofi, C. (eds.) *Proceedings of the 22. Workshop on Foundations of Databases (Grundlagen von Datenbanken)*, Bad Helmstedt, Germany, May 25–28. CEUR Workshop Proceeding, vol. 581 (2010), CEUR-WS.org
14. Jain, R., Molnar, D., Ramzan, Z.: Towards understanding algorithmic factors affecting energy consumption: switching complexity, randomness, and preliminary experiments. In: *Proceedings of the 2005 Joint Workshop on Foundations of Mobile Computing, Workshop on Discrete Algorithms and Methods for MOBILE Computing and Communications*, pp. 70–79. ACM, New York (2005)
15. Koc, H., Ozturk, O., Kandemir, M., Narayanan, S.H.K., Ercanli, E.: Minimizing energy consumption of banked memories using data recomputation. In: *Proceedings of the 2006 International Symposium on Low Power Electronics and Design, ISLPED 2006*, pp. 358–362. ACM, New York (2006)
16. Lafore, R.: *Data Structures and Algorithms in Java*, 2nd edn. SAMS Publishing, Indianapolis (2002)
17. Liveris, N., Zhou, H., Banerjee, P.: A dynamic-programming algorithm for reducing the energy consumption of pipelined system-level streaming applications. In: *Proceedings of the 2008 Conference on Asia and South Pacific Design Automation, ASP-DAC 2008*, Seoul, Korea, pp. 42–48. IEEE Computer Society Press, Los Alamitos (2008)

18. Ozturk, O., Kandemir, M.: Nonuniform Banking for Reducing Memory Energy Consumption. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2005, pp. 814–819. IEEE Computer Society Press, Washington (2005)
19. Potlapally, N.R., Ravi, S., Raghunathan, A., Jha, N.K.: A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols. *IEEE Transactions on Mobile Computing* 5(2), 128–143 (2006)
20. Seddik-Ghaleb, A., Ghamri-Doudane, Y., Senouci, S.-M.: A performance study of TCP variants in terms of energy consumption and average goodput within a static ad hoc environment. In: Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing, IWCMC 2006, pp. 503–508. ACM, New York (2006)
21. Senouci, S.-M., Naimi, M.: New routing for balanced energy consumption in mobile ad hoc networks. In: Proceedings of the 2nd ACM International Workshop on Performance Evaluation of Wireless ad hoc, Sensor, and Ubiquitous Networks, PE-WASUN 2005, Montreal, Quebec, Canada, pp. 238–241. ACM Press, New York (2005)
22. Singh, H., Singh, S.: Energy consumption of TCP Reno, Newreno, and SACK in multi-hop wireless networks. *ACM SIGMETRICS Performance Evaluation Review* 30(1), 206–216 (2002)
23. Sousa, J.P., Balan, R.K., Poladian, V., Garalan, D., Satyanarayanan, M.: User Guidance of Resource-Adaptive Systems. In: Cordeiro, J., Shishkov, B., Ranchordas, A., Helfert, M. (eds.) Proceedings of the Third International Conference on Software and Data Technologies, ICSOFT 2008, vol. SE/MUSE/GSDCA, pp. 36–45. INSTICC Press, Setúbal (2008)
24. Sun, B., Gao, S.-X., Chi, R., Huang, F.: Algorithms for balancing energy consumption in wireless sensor networks. In: Proceeding of the 1st ACM International Workshop on Foundations of Wireless ad hoc and Sensor Networking and Computing, FOWANC 2008, pp. 53–60. ACM, New York (2008)
25. Tuan, T., Kao, S., Rahman, A., Das, S., Trimberger, S.: A 90nm low-power FPGA for battery-powered applications. In: Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays, FPGA 2006, pp. 3–11. ACM, New York (2006)
26. Wang, L., French, M., Davoodi, A., Agarwal, D.: FPGA dynamic power minimization through placement and routing constraints. *EURASIP Journal on Embedded Systems* (1), 7 (2006)

Part V

Knowledge-Based Systems

Analysis of Emergent and Evolving Information: The Agile Planning Case

Rasmus Rosenqvist Petersen and Uffe Kock Wiil

The Maersk Mc-Kinney Moller Institute, University of Southern Denmark
Campusvej 55, 5230 Odense M, Denmark
{rrp, ukwiil}@mmmi.sdu.dk

Abstract. Some information structures are by nature emergent and evolving and as a consequence the retrievable knowledge keeps shifting patterns like a kaleidoscope. Hence, information analysis can be a complex and tedious task. The planning task of agile teams is an example of such a complex information analysis task. In this paper, we present a lightweight planning tool. ASAP is inspired by concepts and principles from spatial hypertext, which have proven successful in supporting information analysis tasks. ASAP runs on a large interactive vertical display on which electronic task cards can be organized into iterations and releases using card hierarchies and separators (a novel visual concept). Several views of the evolving plan are automatically generated to assist the agile team with overviews of tasks, estimates, and assignments. Views are instantly updated to reflect changes to the plan.

Keywords: Information Analysis, Spatial Hypertext, Agile Planning, Electronic Task Cards, Interactive Displays, Automatically Generated Views.

1 Introduction

Some information structures are by nature emergent and evolving and as a consequence the retrievable knowledge keeps shifting patterns like a kaleidoscope [1]. Hence, information analysis can be a complex and tedious knowledge management task. But the fluent nature of the information makes the generated knowledge highly valuable. It can help users make informed decisions by revealing otherwise non-comprehensible or hidden patterns. Spatial hypertext has proven successful in supporting information analysis tasks [2], [3], [4], [5]. The planning task of agile teams is an example of a complex information analysis task that can be supported by spatial hypertext concepts and principles.

Project planning in agile teams is a collaborative process relying on face-to-face communication and shared information to succeed. A commonly used approach to planning involves teams using a large table to plan iterations using paper cards to represent tasks to be carried out. When a planning session is over, the plan is somehow recorded, and the cards are removed from the table. One downside to this is that cards' location on the table and their proximity to other cards may contain important information for the overall plan. When cards are removed from the table, their arrangement is often lost and with it so is the proximity and location information [6].

Our overall goal is to develop a software tool to support agile planning, while preserving the benefits of physical card based planning. Card based planning is suggested by several agile development methods [7] [8] and is used by many agile software teams. During the development of ASAP, we have been working closely with local software companies to identify requirements for the planning tool, to get feedback on suggested features, and to test various versions of the planning tool. A number of important overall requirements have come out of this collaboration:

- A planning tool should support the work of the agile team in a manner that resembles the physical card based approach.
- A planning tool should be lightweight offering only the features that are necessary to solve the task at hand.
- A planning tool should visualize the consequences of the planners' actions allowing them to make informed decisions regarding the plan.

ASAP has been developed to fulfill these overall requirements. It runs on a large interactive vertical display on which electronic task cards can be moved around freely and organized into iterations and releases using card hierarchies and separators (a novel visual concept). Generated plans can be stored, printed, exported, and retrieved again later for future planning sessions.

Once paper cards become electronic, several new opportunities arise. Different views can be automatically generated based on the layout and the attributes of the task cards. ASAP provides overviews of tasks, estimates, status, and assignments. Views in ASAP are instantly updated to reflect the changes made by the planners, thus supporting the planners by visualizing the current status of the plan.

The remainder of the paper is structured as follows. Section 2 introduces the basics of information analysis and spatial hypertext. Section 3 reviews some well-known agile software development methods to identify agile planning practices and techniques. Section 4 looks at existing software support for agile planning. Section 5 presents ASAP, including requirements, design concepts and features, current status and ongoing work, and evaluation. Section 6 briefly describes our approach towards a framework for information analysis based on ASAP. Section 7 summarizes the paper.

2 Information Analysis and Spatial Hypertext

Marshall and Shipman [2] note that information analysts faced with the task of organizing and understanding large amounts of data develop structures over this data over time. As their understanding of the information space changes, the structures they use to characterize the space also change. Systems designed for such analysts are required to support emerging and evolving structures that avoid the problems associated with premature organization and formalization, as discussed, e.g., by Halasz [3] and Marshall et al. [4].

Marshall and Shipman [2] have proposed spatial hypertext to meet these requirements. Spatial hypertext systems allow users to represent pieces of information as visual "icons". Analysts can represent relationships among objects implicitly by varying certain visual attributes (color, size, shape) of the icons and by arranging the icons in arbitrary ways in a two dimensional space.

A spatial parser can then recognize the spatial patterns formed by these icons. Examples of such spatial structures might be lists of red rectangles that contain text or piles of blue ovals that contain images. Both the user and the system can use the structures recognized by this parser to support the task of analysis. For example, the system may recognize some particular type of structure as occurring frequently and conclude that it represents a meaningful abstraction to the analyst. It may then prompt the analyst to recognize this type of structure formally, perhaps by naming it. This formal recognition may allow additional functionality to be provided, such as searches for instances of the structure in the space or replacement of the structure with a new visual symbol that may be “expanded” into its constituent parts.

VKB (Visual Knowledge Builder) [5] is a prominent general purpose spatial hypertext system that supports information analysis as described above.

3 Agile Planning

A number of agile software development practices and techniques focus on and involve planning. We will briefly look at some of these to investigate what types of activities ASAP should be able to support.

The Scrum agile software development method suggests several work practices where planning plays a role: pre-game planning and staging, sprint planning, and the daily meeting [7]. Pre-game planning and staging focuses on identifying desired features which are recorded in the Product Backlog and possibly one or more Release Backlogs. Iterations (sprints) start with two sprint planning meetings where the tasks for the upcoming iteration are planned and estimated. At the daily meeting the sprint plan is reviewed and tasks may end up in the Sprint Backlog.

Extreme Programming (XP) is a well-known agile method that includes 12 core practices – including the Planning Game [7]. The Planning Game is a meeting that occurs once per iteration. The Planning Game is divided into two parts. Release Planning Game focuses on determining what features are included in the next release, and when they should be delivered. Iteration Planning Game focuses on planning the activities and tasks of the developers in the upcoming iteration.

The Crystal family of agile methodologies [8] includes the Blitz Planning technique. Blitz Planning is based on paper task cards laid out on a table surface. The overall idea is to gather the right people, discuss the project details, and end up with a project plan as a result of working through ten predefined steps. Blitz Planning is a variation of the XP Planning Game described above. The two differ in three ways [8]:

- The planning game cards list *user stories*, and the Blitz Planning cards list *tasks*.
- The planning game has the people assume there are no dependencies between stories, while Blitz Planning has people analyze the dependencies between tasks.
- The planning game assumes fixed-length iterations, while Blitz Planning does not assume anything about iteration length.

Another technique included in the Crystal family is the Daily Stand-up Meeting; this is quite similar to Scrum’s daily meeting described above.

The above reviews show that planning is a central activity in agile software development. If we go one step deeper into the practices and techniques, they suggest the use of cards and large surfaces (tables and whiteboards) to create and revise plans.

4 Related Work

Over the last years, many commercial and open source tools have become available to support agile planning. Liu [9] identified three categories of planning support systems: form-based, combined Wiki- and form-based, and board-based.

Form-based. The majority of tools that support agile planning belong to this category. Typical representatives of this category are browser based and provide forms to store a predefined set of information, e.g., effort estimates or priority rankings. Form-based systems provide basic functionality for creating and deleting as well as editing and prioritizing project planning artifacts and can derive supplementary information like total efforts for iterations or remaining work effort from existing data. Existing form-based tools comprise commercial products like Rally [10], VersionOne [11], and ScrumWorks [12] as well as open source products like XPlanner [13].

Wiki- and form-based. Tools like MASE [14] (University of Calgary) take the form-based approach one step further and combine it with another very popular way to share information between people. Users can attach Wiki-pages to stories that can be used to provide additional information related to a task.

Board-based. This category comprises tools like CardMeeting [15], AgilePlanner [9], Distributed AgilePlanner [16], and MasePlanner [17]. A commonality of board-based systems is that they are all mimicking card based planning to a certain extent. They provide ways to create, edit, and delete cards and visually group those cards to indicate relationships. CardMeeting attempts to bridge the gap between browser based systems and physical card based planning. It displays electronic index cards in a web browser. It is primarily focused on the visual aspect of card based planning. It does not provide the iterations and progress tracking that other agile planning tools have. AgilePlanner and its successor Distributed Agile Planner (University of Calgary) are card based tools for collocated and distributed agile planning. Synchronous distributed planning meetings are supported by providing a shared workspace (displayed on vertical displays and/or digital tabletops) for creating, organizing, and editing electronic index cards. Changes made by one team member become visible immediately on connected clients all over the world. MasePlanner (also from University of Calgary) builds on features from MASE and AgilePlanner and is implemented as an Eclipse plug-in with web services for remote connectivity.

ASAP is a board-based agile planning tool supporting collocated agile teams. The ability to support well-known agile practices and techniques such as Crystal Clear's Blitz Planning and XP's Planning Game has played a major role in determining the features of the tool. ASAP is developed to run on large interactive **vertical** displays. Many of University of Calgary's tools have special support for interactive horizontal displays (such as rotation of cards). According to the chosen lightweight strategy, it should be easy (and inexpensive) to run ASAP in existing meeting rooms. Very few meeting rooms have interactive horizontal displays (tabletops), while interactive vertical displays are more common. Commercial tools like the Tool-Tribe Connector (www.tool-tribe.dk) offer a simple, portable, and inexpensive solution that turns any whiteboard into an interactive surface. This allows ASAP to be used in any meeting

room equipped with a whiteboard. The development of ASAP is inspired by techniques from spatial hypertext. This makes the features and interaction in ASAP different from existing board-based tools. Task cards are easy to create, manipulate, and organize. Besides the traditional spatial organization of task cards on a surface, ASAP provides two additional organization features – a hierarchical view of the organization of tasks and subtasks and a novel visual concept (the separator) used to separate and group task cards.

5 The ASAP Approach

This section presents ASAP including requirements, design concepts and features, current status and ongoing work, and evaluation.

5.1 Requirements

Based on interactions with local software companies three overall requirements for agile planning tools were identified:

- *A planning tool should support the work of the agile team in a manner that resembles the physical card based approach.* A computer tool like ASAP should not alter a workflow that works. It should simply support the existing workflow (in this case board-based planning) and, if possible, provide additional support for the individual steps in the workflow enabling the user to perform the tasks better and/or faster.
- *A planning tool should be lightweight offering only the features that are necessary to solve the task at hand.* There are many examples of tools that provide a lot more features than necessary to solve a given task. Let us consider Microsoft Word. Typical Word users only make use of a very limited subset of the features in Word (say 10 %) to solve most of their writing tasks (say 90 %). This can result in complex tools that are difficult to use. The entry barrier becomes higher for new users. The overhead of using the tool may outweigh the benefits of the tool. ASAP goes the other way and provides only the most essential features for agile planning. A lightweight planning tool that is intuitive and easy to use will result in a much lower entry barrier for new users and will be able to support most of their planning needs.
- *A planning tool should visualize the consequences of the planners' actions allowing them to make informed decisions regarding the plan.* The agile team is in charge of the planning process. The tool supports the team by using its computing power to instantly visualize the consequences of the individual steps in the planning process.

In particular, two local software companies have contributed to the generation of the above overall requirements. Mikro Værkstedet (www.mikrov.dk) is a small software development company (<50 employees) focusing on educational software. KMD (www.kmd.dk) is a large software development company (3000+ employees) focusing primarily on software for the public sector.

Formal interviews have been conducted with agile software development team managers at both places regarding requirements for an agile planning tool. Earlier versions of ASAP were discussed with the same team managers to focus the development of ASAP on the most essential features. Finally, the team managers are currently using ASAP in ongoing software development projects using agile practices. Feedback from the use has been collected and ASAP has been enhanced based on this (see Section 5.4).

A set of functional requirements for ASAP have been derived based on the overall requirements and the desire to support existing agile planning practices and techniques. These requirements are not surprisingly somewhat overlapping with the agile planning requirements presented by Liu [9] and Morgan [16].

1. **Supporting Agile Planning Objects.** Creating, editing, and deleting task cards are core agile planning activities. We have adopted the Crystal Clear Blitz Planning notation of *tasks* instead of *user stories*.
2. **Organizing Agile Planning Objects.** The ability to move task cards around freely and organize them into iterations and releases are core agile planning practices.
3. **Supporting Multiple Iterations.** Agile teams should be able to make both short term planning (next iteration) and long term planning (future iterations and releases).
4. **Supporting Hierarchies of Planning Objects.** Breaking down tasks into subtasks is a well-known strategy for handling complexity (divide and conquer). Hierarchies also provide a way to handle large projects with many tasks (addressing the scalability issue).
5. **Visualizing Consequences of Planning Actions.** Visualizing the consequences of the planners' actions allows them to make informed decisions regarding the plan.
6. **Supporting Estimation and Tracking** [18]. Adding estimates to tasks cards allows the agile team to get an overview of the duration of iterations. Adding task status to Task Cards allows the agile team to track the status of single tasks as well as iterations.
7. **Managing Team Members and Resources.** Assigning tasks to team members allows agile teams to plan their resources.
8. **Re-using Experiences from Past Planning Sessions.** Access to old planning sessions allows the team members to include past experiences in their current planning sessions.

We consider requirements 1 through 5 to be essential to support agile planning the ASAP way. Requirements 6 through 8 are also important, but according to the chosen strategy they will be provided in a lightweight manner.

5.2 Design Concepts and Features

The concepts used in ASAP are developed based on the three overall and eight functional requirements listed above. Figure 1 illustrates a planning session with physical cards laid out on a table.

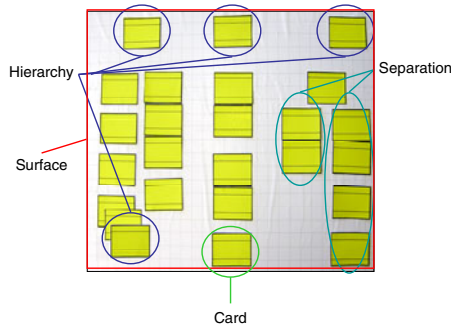


Fig. 1. A planning session with physical cards on a table

Several visual features can be observed in the figure: The use of task cards, the large surface used to organize the cards, the organization of cards into hierarchies, the visual separation of cards, the location of cards, the proximity of cards, etc. These observations have influenced the design of concepts and features in ASAP.

The design of the user interface in ASAP is based on well-known interaction principles (such as direct manipulation, drag and drop, and cut and paste), interaction features (such as menus, toolbars, and shortcut icons), and interaction metaphors (Windows Explorer style hierarchies and Windows desktop style surface).

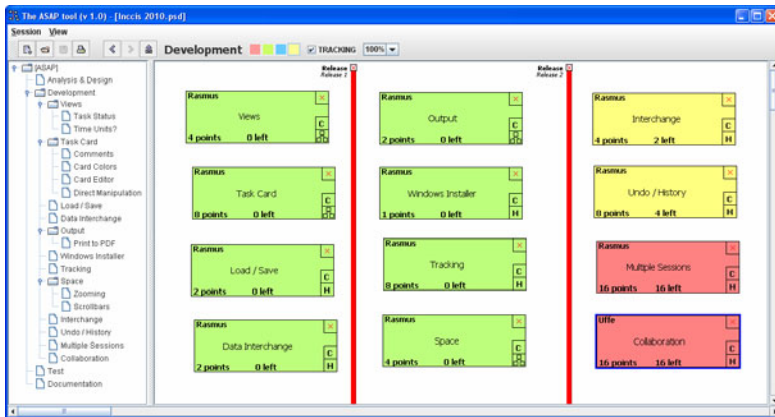


Fig. 2. The main window in ASAP

Figure 2 provides a screenshot of the main window in ASAP consisting of three parts – two views as well as the menus and toolbar at the top. The View to the left provides an overview of the task hierarchy using an interaction metaphor similar to the one used in Windows Explorer. The View to the right provides a large surface (Space) on which task cards can be organized freely using an interaction metaphor similar to the one used by the desktop in Windows. Each central design concept and tool feature of ASAP is explained below.

The **Space** (right hand View in Figure 2) is a well-known concept adopted from spatial hypertext. A Space in ASAP is a large two dimensional surface used to organize electronic task cards. The **Hierarchy** (left hand View in Figure 2) provides a tree overview of the organization of tasks and subtasks. The tree root reflects the name of the planning session, nodes in the tree are tasks containing subtasks, and leafs in the tree are tasks with no subtasks. The Hierarchy View and the Space View are synchronized in the sense that changes made in one View are instantly reflected in the other View. There are no limitations to the number of nested hierarchies. The two Views are separated by a divider that can be moved left or right to expand/minimize the Views depending on the users' preference. A zooming feature on the toolbar allows the user to view cards in the space in varying sizes (50%, 75%, 100%, and 150%). The default size is 100%.

The **Task Card** is the basic planning object used in ASAP. It represents the paper equivalent. It is overlaid with a grid (3 by 3 cells) in which each cell is assigned specific meaning. The value of cells can be changed by the user. Task Cards are created by making a dragging gesture inside a Space that resembles the shape of a Task Card. A Task Card is deleted when pressing the "red cross" in the upper right hand corner. The icon in the lower right hand corner is used to create and traverse hierarchies. The icon can have two different forms. A card with no subtasks is depicted with an icon similar to an "H". If the user clicks the "H" icon, then a new subspace is created and opened. The user can now add subtasks in the subspace. A subspace offers exactly the same functionality as a Space. A card with subtasks is depicted with a different icon in the lower right hand corner. If the user clicks this icon, then the subspace opens allowing the user to manipulate the subtasks. We use the term Space to cover both the top level Space and the subspaces. The "C" icon is used to manage comments relevant to the task card (in a separate window). Task Cards can have different colors (toolbar option).

Task Cards can be configured using a **Card Editor**. Since the optimal Task Card design may differ depending on the project team and planning task, a Card Editor allows for configuration (personalization) of the Task Card layout. By default the cell in the upper left hand corner is named Person and is of type [PERSON], the cell in the lower left hand corner is named Estimate and is of type [TIME], the cell in the middle is named Task and is of type [TASK], and the cell below the Task cell is named Tracking and is of type [TIME]. The Person, Estimate, and Tracking cells are used for estimates and tracking and for handling team members and resources. The Tracking cell is only visible when tracking is turned on (toolbar option).

A **Separator** (red vertical bar) is used to group vertically dependent Task Cards visualizing the horizontal separation of dependencies (e.g., a time line). Separators are created by making a dragging gesture inside a Space that resembles the shape of a Separator. Separators can be assigned a name (description), a date, and a type. Currently, the following types of Separators exist: Separation, iteration, walking skeleton, release, and milestone. The Separator is a novel visual structuring concept that is introduced to support the planning task by visualizing dependencies and groupings among Task Cards.

The Space is immediately parsed by a **Spatial Parser** every time a change occurs. The algorithms parsing the Separator(s) sort them according to their position, and then apply methods to detect Task Cards to the left, to the right, and between two

Separators. Spatial parsing is done along the y-axis clarifying vertical dependencies. The algorithms parse the Task Cards grouped together by Separators, according to their y-coordinates. The Card with the lowest y value is identified as the first task in the grouping (e.g., iteration) and so forth. The concept of a spatial parser is a well-known concept adopted from spatial hypertext. The Spatial Parser automatically generates **Views** that are relevant to the planning task. The generated Views visualize the consequences of the planners' actions. The Spatial Parser can currently generate two types of Views.

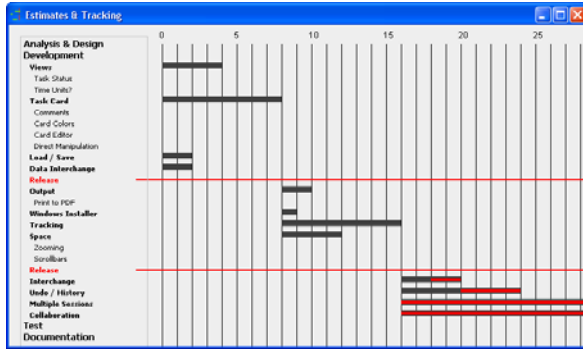


Fig. 3. The auto-generated Estimates & Tracking View

The Estimates & Tracking View (Figure 3) is auto-generated based on the layout of Task Cards in the Space (and subspaces), the time estimates of Task Cards, and the position of Separators in the Space (and subspaces). When a Task Card or Separator is moved or when a time estimate is changed, the Estimates & Tracking View is immediately updated to show the revised Plan. The time line at the top indicates the number of planning units (hours, half days, or days) assigned to a task. The Estimate cell stores the assigned time estimate of a task, while the Tracking cell stores the estimated effort needed to complete the task. The grey indicators show estimates of tasks and the red indicators show the remaining effort needed to complete the tasks (only visible when tracking is turned on).

The Team Members & Resources View (Figure 4) is auto-generated based on the cells of type PERSON and TIME in the Task Card. The Team Members & Resources View allows the names of team members to be added. Once names are added, each task can be assigned to a team member by updating the cell of type PERSON. The View also shows the estimate of each task as well as the total estimate of the tasks assigned to a person. The Team Members & Resources View is instantly updated when a task has been assigned to a team member and when an estimate has been changed.

A **Planning Session** is something that is not normally finished in one go; most often, there is a need to revise an existing plan. This implies the need for storing Planning Sessions and continuing (revising) them at a later time. ASAP allows Planning Sessions to be stored and opened again later. ASAP defines its own file type (*.psd – planning session data) for this purpose. This functionality is available both in the “Session” menu and on the toolbar through shortcut icons.

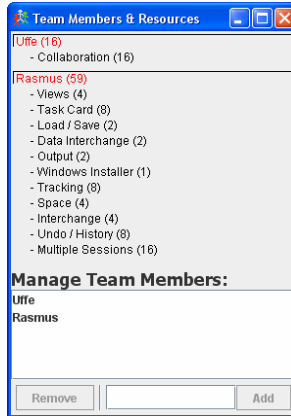


Fig. 4. The auto-generated Team Members & Resources View

Plans can also be **exported** to XML and CSV (comma separated value) formats. Visual objects are serialized into an XML string which can be saved in a text file. The XML format contains information about Separators and Task Cards including a unique ID, cell information, card location, card size, color, font size, and other font information. Information about cells, their type and location is also stored. The CSV format includes the values of the name, estimate, tracking, and person cells of all cards. The use of XML and CSV as interchange formats makes it possible to transform the plan generated in ASAP into formats that can be imported by other agile planning tools or third-party project management applications.

Finally, plans can be **printed** – either on paper or to a PDF file. It is possible to print a single View (i.e., current Space or Estimates & Tracking View). A full print of the plan consists of the following parts:

- An overview of the Task Cards in each Space – the hierarchy (tree structure) is traversed and printed in a depth-first manner: top Space, all subspaces of the first Task Card, all subspaces of the second Task Card, etc.
- An overview of the Estimates (as seen in the Estimates & Tracking View).
- An overview of the Team Members and their assignments (as seen in the Team Members & Resources View).
- A page for each team member with a list of assigned tasks.

5.3 Current Status and Ongoing Work


The current version of ASAP (September 2010) fulfills most of the identified functional requirements for an agile planning tool:

Requirements 1-6: Fully supported. All the desired features are supported.

Requirement 7: Partly supported. Management of team members is supported. Management of resources is not yet supported.

Requirement 8: Partly supported. To some extent, this feature is already available in the current version. It is possible to load an old planning session into ASAP and reuse it as a starting point for a new planning session. In this way, experiences from an earlier project can be reused.

Our future work plan involves the following features:

- **Management of resources.** We plan to add lightweight support for management of resources.
- **Multiple sessions.** We plan to add an additional level of support allowing reuse from multiple past planning sessions.
- **Interchange.** We are currently testing how well the provided XML and CSV formats interface to existing project management tools (i.e., Microsoft Project) as well as existing commercial agile planning tools.
- **History.** We are currently implementing a history feature similar to the one available in VKB [5]: 

This feature will support unlimited undo of actions as well as scrolling back in history and possibly exploring alternate planning steps.

- **Collaboration.** Currently, we only provide support for collocated meetings. We plan to provide support for distributed planning meetings also.

5.4 Evaluation

ASAP has been developed to support agile planning practices and techniques from well-known software development methods. We have previously shown [19] that an earlier version of ASAP (January 2008) can support the activities of the Blitz Planning technique from the Crystal family [8]. The current version is developed to provide support for a broader set of planning tasks as specified in the software development methods reviewed in Section 3.

ASAP is developed as a lightweight planning tool providing only the most essential features for agile planning. Features are provided in an independent manner that allows the planners to only use the features that they need for the task at hand. Features in ASAP can be divided into two categories: basic features provided by the Space and Hierarchy Views and additional features provided by the Estimates & Tracking and the Team Members & Resources Views. Planners engaged in simple planning tasks may only need the basic features provided by the Space View. More complex planning tasks require the use of additional Views. Thus, planners only “pay” for what they use in terms of tool complexity. There is no additional overhead for features that are not being used.

ASAP is currently being used and evaluated in software projects conducted at local software companies. Figure 5 depicts a typical set up of ASAP in planning sessions at the local software companies. The main window of ASAP (providing the Space and Hierarchy Views) is displayed on a large interactive surface (e.g., a SMART Board™) allowing the software team members to jointly plan and interact with the planning objects. The additional Views (Estimates & Tracking and Team Members & Resources) allowing the team members to instantly see the consequences of their actions are displayed on a separate screen.

The purpose of the evaluations is to explore the borders of the applicability of our lightweight approach to agile planning. Projects are monitored with respect to project size and the required feature intensity. We wish to find out if there is a limit to the

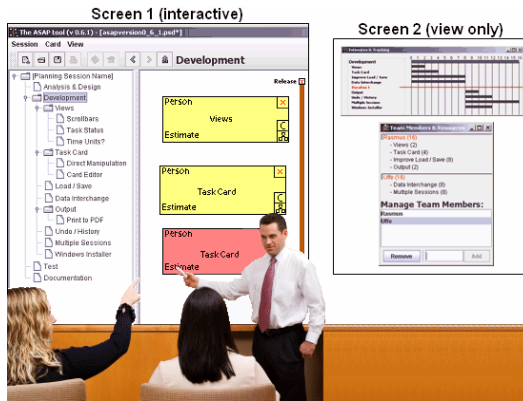


Fig. 5. Typical use of ASAP in a planning session

size of projects that ASAP can support (primarily in terms of numbers of tasks). We also wish to find out what features are being used in what types of projects. Finally, we wish to find out if the features provided by ASAP are sufficient for most agile planning purposes. Evaluations are still ongoing. However, the results and feedback is positive and encouraging as indicated below by statements from end users:

- “It is a good idea to base the interaction in ASAP on well known concepts. The tool is intuitive and easy to use.”
- “Plans are faster to make with ASAP than our previous method. ASAP supports an effective planning process.”
- “We believe that the plans generated with ASAP are better than our usual plans. Plans are easy to store, revise, and distribute.”
- “We always keep an overview of the plan visible in the project room. It helps the team members in their daily work.”

Thus, we are confident that the chosen lightweight strategy turns out to be successful. Suggestions for improving the tool received from companies have already resulted in enhancements of existing features (export and tracking features) and addition of new features (card colors and zooming feature). A demo version of ASAP can be downloaded for evaluation from www.planitnow.dk.

6 Towards a Framework for Information Analysis

We are currently developing a framework for information analysis based on ASAP. The framework contains generic information analysis and software tool functionality in its basic framework components (Figure 6). These components can be used to support specific information analysis tasks from various application domains. So far the framework has been used to develop applications for two domains: agile planning (ASAP) and intelligence analysis (the CrimeFighter Investigator tool).

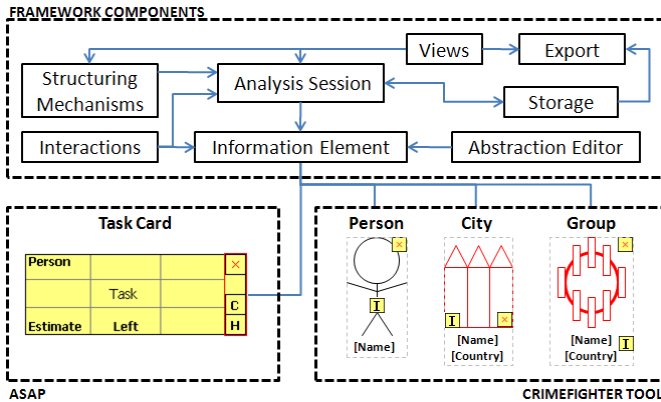


Fig. 6. A framework for information analysis and two application domains

7 Conclusions

The ASAP approach to agile planning has been developed based on different types of analysis work:

- **Involving End Users.** We have interacted with software companies that practice agile planning to get their input.
- **Exploring Methods.** We have explored planning practices and techniques from several agile software development methods.
- **Studying Related Work.** We have found inspiration from existing agile planning tools.

Together, this resulted in three overall and eight functional requirements that have guided the development. Currently, most of the envisioned features are supported and the tool is being used and evaluated by local software companies.

ASAP is inspired by previous work on the use of spatial hypertext to support the knowledge management task known as information analysis. ASAP is based on the Construct Space Tool [20]. The work has so far resulted in three main contributions:

- We have developed a board-based agile planning tool to help software teams plan their projects. The tool offers only the features that are necessary to solve the task at hand according to the chosen lightweight strategy.
- Agile planning is viewed as a complex knowledge management task (information analysis). Specific features have been developed to support the planning requirements. The work has resulted in novel ideas such as the visual Separator concept, the Hierarchy View, and the use of the Spatial Parser to auto-generate Views that are relevant to the planning process.
- A novel framework for supporting information analysis tasks has evolved from the work on ASAP. The framework is currently being used to implement support for information analysis tasks related to intelligence analysis.

We believe that the inspiration from several fields combined with the chosen lightweight approach has resulted in a tool that is well suited for agile planning.

Acknowledgements. This paper is an extended and revised version of a paper previously published at the International Conference on Software and Data Technologies (ICSOFT 2009) [21].

References

1. Hjørland, B., Albrechtsen, H.: Toward a New Horizon in Information Science: Domain-Analysis. *Journal of the American Society for Info. Science* 46(6), 400–425 (1995)
2. Marshall, C.C., Shipman, F.M.: Spatial hypertext: Designing for Change. *Communications of the ACM* 38(8), 88–97 (1995)
3. Halasz, F.: Reflections on NoteCards: Seven issues for the Next Generation of Hypermedia Systems. *Communications of the ACM* 31(7), 836–852 (1988)
4. Marshall, C.C., Shipman, F.M., Coombs, J.H.: VIKI: Spatial Hypertext Supporting Emergent Structure. In: *Proceedings of the European Conference on Hypermedia Technologies (ECHT 1994)*, Edinburgh, Scotland, pp. 13–23 (1994)
5. Shipman, F.M., Hsieh, H., Maloor, P., Moore, J.M.: The Visual Knowledge Builder: A Second Generation Spatial Hypertext. In: *Proceedings of Hypertext 2001*, Aarhus, Denmark, pp. 113–122. ACM Press, New York (September 2001)
6. Morgan, R., Walny, J., Kolenda, H., Ginez, E., Maurer, F.: Using Horizontal Displays for Distributed & Collocated Agile Planning. In: *Concas, G., Damiani, E., Scotto, M., Succu, G. (eds.) XP 2007. LNCS, vol. 4536, pp. 38–45. Springer, Heidelberg (2007)*
7. Larman, C.: *Agile & Iterative Development – A Manager’s Guide*. Addison-Wesley, Reading (2004)
8. Cockburn, A.: *Crystal Clear – A Human-Powered Methodology for Small Teams*. Addison-Wesley, Reading (2005)
9. Liu, L.: *An Environment for Collaborative Agile Planning*. M.Sc. Thesis, Department of Computer Science, University of Calgary (2006)
10. Rally Software, *Rally’s Agile Lifecycle Management Solutions*, (2009), <http://www.rallydev.com/products.jsp>
11. VersionOne, *Agile Project Management Tools* (2009), <http://www.versionone.com/products.asp>
12. Danube, *ScrumWorks* (2009), <http://www.danube.com/scrumworks/basic>
13. XPlanner, *XPlanner Overview* (2009), <http://www.xplanner.org/index.html>
14. Maurer, F.: Supporting Distributed Extreme Programming. In: *Wells, D., Williams, L. (eds.) XP 2002. LNCS, vol. 2418, pp. 13–22. Springer, Heidelberg (2002)*
15. CardMeeting, *CardMeeting* (2009), <http://www.cardmeeting.com>
16. Morgan, R.: *Distributed AgilePlanner: A Card Based Planning Environment for Agile Teams*. M.Sc. Thesis, Department of Computer Science, University of Calgary (2008)
17. Morgan, R., Maurer, F.: *MasePlanner: A Card-Based Distributed Planning Tool for Agile Teams*. In: *Proceedings of ICGSE 2006, Florianopolis, Brazil, pp. 132–138. IEEE Computer Society Press, Los Alamitos (2006)*
18. Cohn, M.: *Agile Estimation and Planning*. Prentice-Hall, Englewood Cliffs (2006)
19. Petersen, R.R., Wiil, U.K.: ASAP: A Planning Tool for Agile Software Development. In: *Proceedings of Hypertext 2008, Pittsburgh, PA, pp. 27–32. ACM Press, New York (June 2008)*
20. Wiil, U.K., Hicks, D.L.: Tools and Services for Knowledge Discovery, Management and Structuring in Digital Libraries. In: *Proceedings of CE 2001, Anaheim, CA, pp. 580–589 (August 2001)*
21. Petersen, R.R., Wiil, U.K.: ASAP: A Lightweight Tool for Agile Planning. In: *Proceedings of the Fourth International Conference on Software and Data Technologies (ICSOFT 2009)*, Sofia, Bulgaria, pp. 265–272. INSTICC Press (July 2009)

Emotion Based User Interaction in Multimedia Educational Applications

Efthymios Alepis¹, Maria Virvou¹, and Katerina Kabassi²

¹ Department of Informatics, University of Piraeus
80 Karaoli & Dimitriou St., 18534, Piraeus, Greece
{talepis,mvirvou}@unipi.gr

² Department of Ecology and the Environment, Technological Educational Institute of the
Ionian Islands, 2 Kalvou Sq., 29100 Zakynthos, Greece
kkabassi@unipi.gr

Abstract. Towards building an affective educational system we explore the human-computer interaction that is based on two modalities, namely the computer's keyboard and the computer's microphone. The resulting system is affective by having the ability to recognize human emotional signals, as well as by being able to generate emotions for pedagogical educational reasons. Both for the recognition and for the generation of emotions within the educational environment we have incorporated two sophisticated theories that are used as mechanisms for human-machine emotional interaction. The first mechanism emerges from a well known decision making theory, while the second mechanism uses quite recent findings in the area of cognitive psychology. In our research, we have investigated the recognition of emotions from the above mentioned modalities with respect to six basic emotional states, namely happiness, sadness, surprise, anger and disgust as well as the emotion-less state which we refer to as neutral. The purpose of the incorporation of the supplementary emotional interaction between users and computers was to assist the educational processes in e-learning applications and also to support the instructors' work during their authoring duties.

Keywords: E-learning, affective interaction, bi-modal interaction, generation of emotions, OCC theory, decision making theories, educational agents.

1 Introduction

One of the most important challenges in educational software is to produce e-learning and tutoring systems that can be characterized as intelligent. Incorporating "intelligence" in educational systems may enhance the whole learning process and also make the interaction between users and computers more friendly and profitable. Perceiving, learning and adapting to the world around us are commonly labeled as intelligent behavior [1]. In many situations human-computer interaction may be improved by multimodal emotional interaction in real time [2], [3]. Affective computing has recently become a very important field of research because it focuses on recognizing and reproducing human feelings within human computer interaction.

Human feelings are considered very important but only recently have started being taken into account in software user interfaces. Thus, the area of affective computing is not yet well understood and needs a lot more research to reach maturity.

In the last decade, education has benefited a lot from the advances of Web-based technology. Indeed, there have been many research efforts to transfer the technology of ITSs and authoring tools over the Internet. Past reviews [4, 5] have shown that all well-known technologies from the areas of ITS have already been re-implemented for the Web. Some important assets include platform-independence and the practical facility that is offered to instructors of authoring e-learning courses at any time and any place. However, this independence from real instructors and classrooms may cause emotional problems to students who may feel deprived of the benefits of human-human interaction. This may affect the educational process in a negative way. A remedy for these problems may lie in rendering human-computer interaction more human-like and affective in educational software. To this end, the incorporation of speaking, animated educational agents in the user interface of the educational application can be very important.

Indeed, the presence of animated, speaking educational agents has been considered beneficial for educational software [6, 7]. Instructors that may use educational authoring tools should not necessarily be computer experts and should be helped to develop sophisticated educational applications in an easy and cost-effective way [8]. Affective computing may be incorporated into sophisticated educational applications by providing adaptive interaction based on the user's emotional state. Regardless of the various emotional paradigms, neurologists/psychologists have made progress in demonstrating that emotion is at least as and perhaps even more important than reason in the process of decision making and action deciding [9]. Moreover, the way people feel may play an important role in their cognitive processes as well [10].

As Picard points out that one of the major challenges in affective computing is to try to improve the accuracy of recognizing people's emotions [11]. Ideally, evidence from many modes of interaction should be combined by a computer system so that it can generate as valid hypotheses as possible about users' emotions. It is hoped that the multimodal approach may provide not only better performance, but also more robustness [1].

In previous work, the authors of this paper have implemented and evaluated with quite satisfactory results from the users' perspective, other educational systems with emotion recognition capabilities [12]. As a next step we have extended our affective educational system by employing fully programmed educational agents that are able to express a variety of emotions.

Educational agents may be parameterized in many aspects, the way they speak, the pitch, speed and volume of their voice, their body-language, their facial expressions and the content of their messages. Educational agents are able to express specific pedagogical emotional states by the incorporation of the OCC [13] model. The resulting educational system incorporates an affective authoring module that relies on the OCC theory. The system uses the OCC cognitive theory of emotions for modelling possible emotional states of users-students and proposing tactics to the instructors for improving the interaction between the educational agent and the student while using the educational application. Through the incorporation of the OCC model, the system may suggest that the tutoring educational agent should express a specific emotional state to the student for the purpose of motivating her/him

while s/he learns. Consequently, the educational agent may become a more effective instructor, reflecting the instructors' vision of teaching behaviour.

However, as yet there are no authoring tools that provide parameterization in user interface components such as speech-driven, animated educational agents. The present educational system provides the facility to authors to develop tutoring systems that incorporate speaking, animated emotional agents who can be parameterized by the authors-instructors in a way that reflects their vision of teaching behaviour in the user interface of the resulting applications.

2 Overview of the System

The resulting system supports either the installation of a standalone application, or alternatively the installation of a client module. As a result, the educational application can be installed either on a public computer where both students and instructors have access, or alternatively each student may have a copy on his/her own personal computer. The underlying reasoning of the system is based on the student modelling process of the educational application. The system monitors and records all students' actions while they use the educational application and tries to diagnose possible problems, recognise goals, record permanent habits and errors that are made repeatedly. Adaptive help is provided through the tutoring agents that not only support the students' educational process, but also interact affectively with the students by expressing emotional states for pedagogical purposes. The incorporated model that controls the tutoring agents' behavior is described in section 4. The inferences made by the system concerning the students' characteristics are recorded in their student model. Hence, the system offers advice adapted to the needs of individual students. The system's database is used to hold all the necessary information that is needed for the application to run and additionally to keep analytical records of the performance of all the students that use the educational application.

While using the educational application from a desktop computer, students are able to retrieve information about a particular course. In the example of Figure 1 a student is using the e-learning system for a medical course about anatomy. The information is given in text-form while at the same time an animated agent reads it using a speech engine. Students may choose specific parts of the theory and the available information is retrieved from the system's database. Both each course's data as well the way these data are presented to the users (orally or acoustically) is controlled by the authoring "back-stage" module that is operated by the instructors of each e-learning course. Each user has privileges in specific parts of the educational system and each user's rights are determined by the systems user modeling mechanism. As an example, users as students have access to the forms of the educational application that are illustrated in figures 1 and 2, while their instructors have also supplementary access to the forms of the educational application that are illustrated in the next section in figures 4 and 5. As it is illustrated in figure 2, students are able to take tests that include questions, answers, multiple-choice, etc, concerning specific parts of the theory. The tutoring agent is also present in these modes, in order to make the interaction more human-like and to assist the student by providing pedagogical assistance when it is needed.

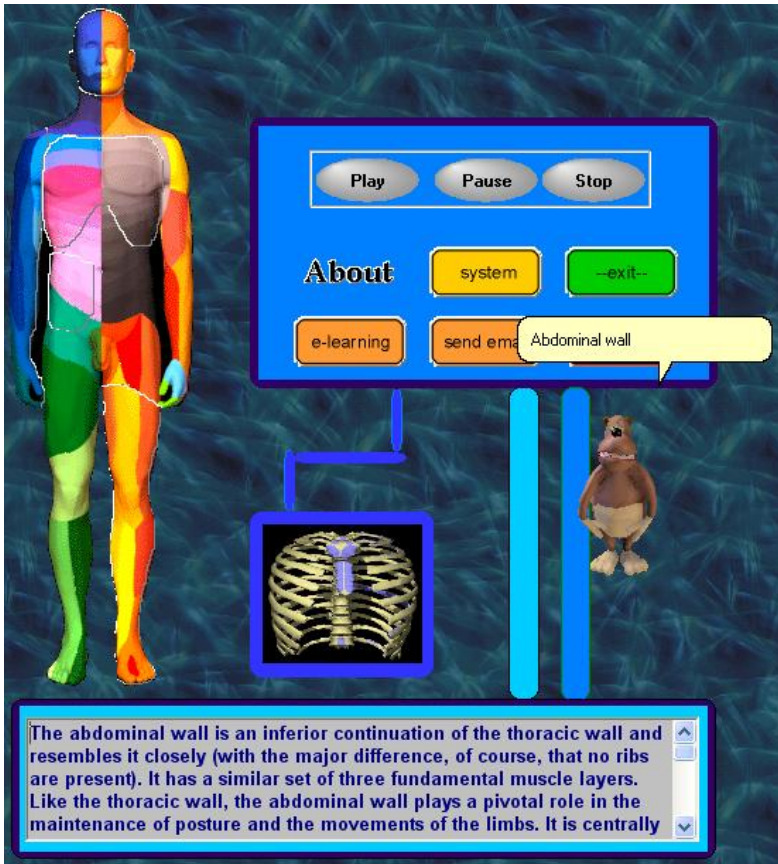


Fig. 1. The educational application with the presence of a tutoring agent

For example the tutoring agent may whisper a help tip to a student if the system determines that this student is confused. Alternatively, if a student makes a small spelling mistake (while s/he obviously knows the correct answer to a question), the animated agent may prompt this student to check his/her spelling. The agent's entire interaction is emotional, which means that the agent is expected to behave in a way that expresses basic emotional states, such as happiness, boredom, fear and surprise. Correspondingly, the agents' behavior incites the users of the educational application to express themselves emotionally and freely and thus give more evidence to the system's emotion recognition module.

3 Educational Agents

In this section we describe the animated agents that are incorporated in the educational system. The tutoring courses that result from the authoring process described in this paper incorporate animated tutoring agents that act as tutors within



Fig. 2. A snapshot of the educational application while a user is taking a test



Fig. 3. Animated affective agents

the educational environment. All the animated agents are fully programmable and have the important ability to act emotionally. The agents can move around the tutoring text and can show parts of the theory in real time (Figure 3). They also incorporate features of human body-language, such as gestures, facial expressions and special movements. The educational agents may show patience while the students read theory for a certain period of time, boredom if the student is not responding to the system, wonder if the student makes an unexpected move, etc. The agent's behaviour is programmatically controlled by an underlying mechanism that relies on the OCC theory, described in the next section.

Instructors may choose from 27 available speech engines that the system incorporates. These speech engines are synthesisers that produce different voices. The system also offers the facility of parameterising these voices by changing the pitch, speed and volume, as illustrated in figure 4. Thus, the resulting tutoring system may use the voices differently in different contexts to show enthusiasm, when the student

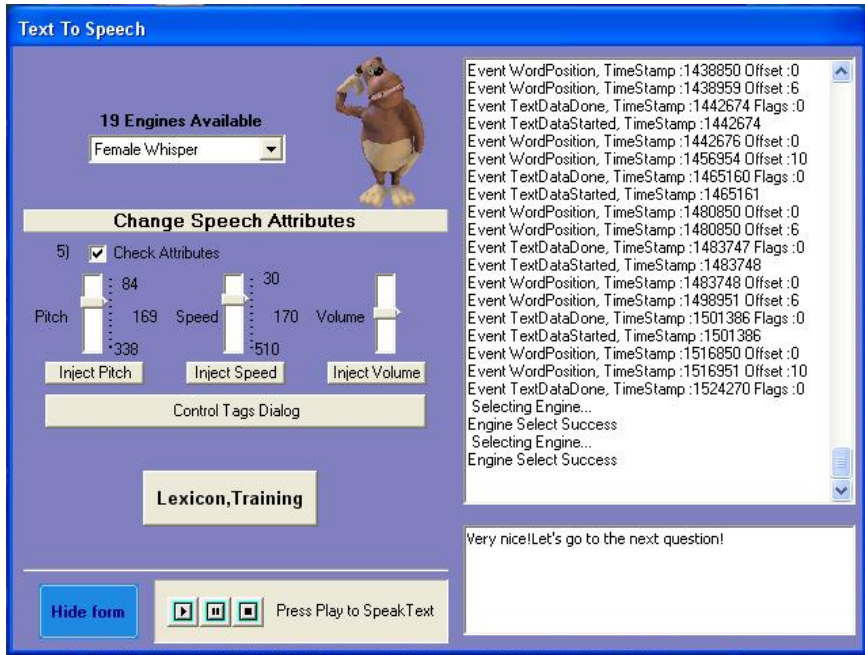


Fig. 4. Setting parameters for the voice of the tutoring agent

is doing particularly well, to imitate whisper, when it judges that the student needs help, or even to show anger when the student is consistently careless and does not pay any attention to the educational system.

In order to produce an “angry” tone of speaking for the animated agent, as an example, instructors may increase the pitch the speed and the volume of the speech engine. This may also be achieved by selecting an appropriate speech engine from the ones that are available. Additionally, the instructor may use the form illustrated in Figure 5 that provides pre-installed voice tones for the agents.

The system incorporates built in tools, to which only the instructors have access. These tools help the instructors modify the behavior of the characters further, with the agents’ emotion generation facility as the final objective. Not only can the instructor command the assistant to say something under certain circumstances, but s/he can also add commands in the text that will be spoken, in a way that the agent may seem to express a specific emotional state. These commands are understood by the system and are interpreted into changing speech attributes, body movements, facial expressions, etc.

4 Emotion Recognition and Emotion Generation

4.1 Recognizing Emotional States

For emotion recognition purposes, a user monitoring component has been used to capture all user input data during the interaction with the educational application. The

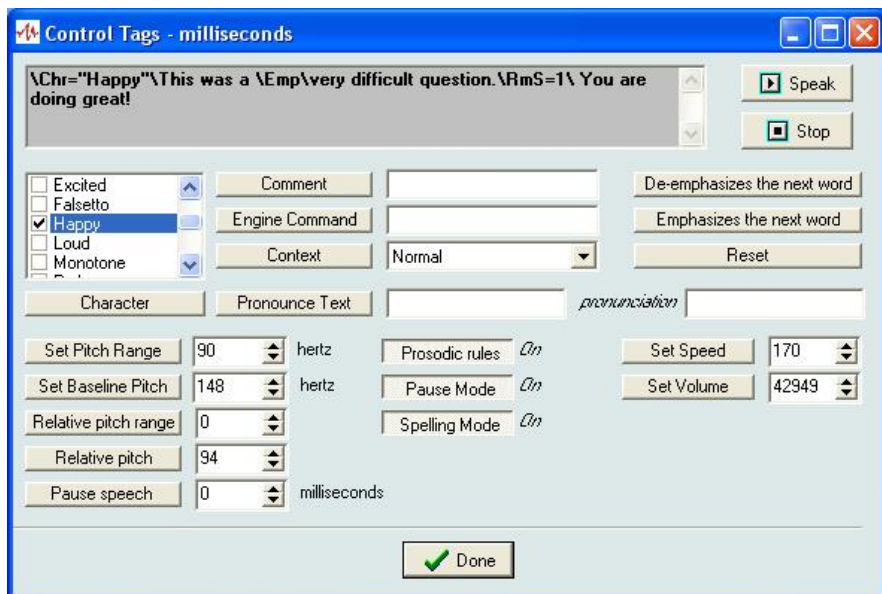


Fig. 5. Pre-installed tones for the voice of the tutoring character

monitoring component is illustrated in figure 6. Input data consist of audio information that has been collected through the keyboard, as well as audio information that has been collected through the microphone.

The analysis of the data collected by the monitoring component, revealed some statistical results that associated user input actions through the computer's keyboard and microphone with possible emotional states of the users. More specifically, considering the keyboard we have the following categories of user actions: a) user types normally b) user types quickly (speed higher than the usual speed of the particular user) c) user types slowly (speed lower than the usual speed of the particular user) d) user uses the "delete" key of the keyboard often e) user presses unrelated keys on the keyboard f) user does not use the keyboard.

Considering the users' basic input actions through the computer's microphone we have 7 cases: a) user speaks using strong language b) users uses exclamations c) user speaks with a high voice volume (higher than the average recorded level) d) user speaks with a low voice volume (low than the average recorded level) e) user speaks in a normal voice volume f) user speaks words from a specific list of words showing an emotion g) user does not say anything.

Therefore, whenever an input action is detected the system records a vector of input actions through the keyboard (k1, k2, k3, k4, k5, k6) and a vector of input actions through the microphone (m1, m2, m3, m4, m5, m6, m7).

All the above mentioned attributes are used as Boolean variables. In each moment the system takes data from the bi-modal interface and translates them in terms of keyboard and microphone actions. If an action has occurred the corresponding attribute takes the value 1, otherwise its value is set to 0. Therefore, for a user that speaks with a high voice volume and types quickly the two vectors that are recorded

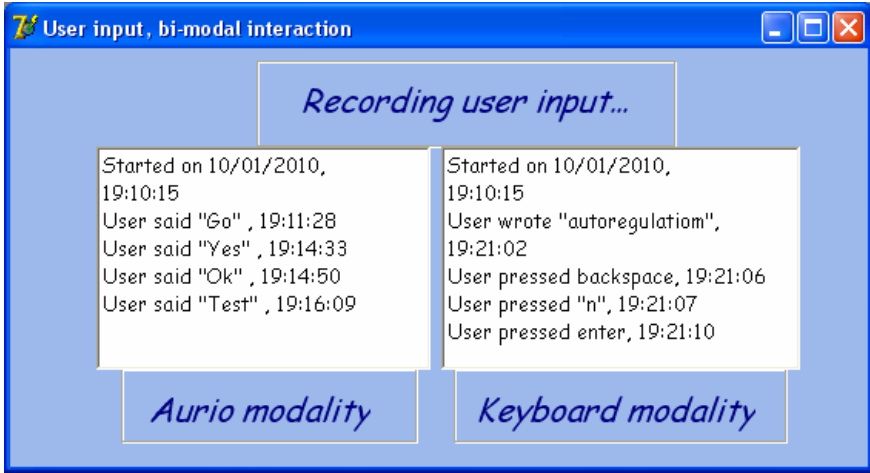


Fig. 6. Snapshot of operation of the user modelling component

by the system are: $k = (0, 1, 0, 0, 0, 0)$ and $m = (0, 0, 1, 0, 0, 0)$. These data are further processed by the decision making model for determining the emotion of the user.

A previous empirical study revealed the attributes that are taken into account when evaluating different emotions [14]. However, these attributes were not equally important for evaluating different emotions. In this study human experts resulted that one input action does not have the same weight while evaluating different emotions. Therefore, the weights of the attributes (input actions) were calculated in order to be used by the decision making model.

For the evaluation of each alternative emotion the system uses SAW [15, 16] for a particular category of users. According to SAW, the multi-attribute utility function for each emotion in each mode is estimated as a linear combination of the values of the attributes that correspond to that mode.

The SAW approach consists of translating a decision problem into the optimisation of some multi-attribute utility function U defined on A . The decision maker estimates the value of function $U(X_j)$ for every alternative X_j and selects the one with the highest value. The multi-attribute utility function U can be calculated in the SAW method as a linear combination of the values of the n attributes:

$$U(X_j) = \sum_{i=1}^n w_i x_{ij} \tag{1}$$

where X_j is one alternative and x_{ij} is the value of the i attribute for the X_j alternative.

In view of the above, for the evaluation of each emotion taking into account the information provided by the keyboard is done using formula 2.

$$em_{ke_1} = w_{e_1k_1}k_1 + w_{e_1k_2}k_2 + w_{e_1k_3}k_3 + w_{e_1k_4}k_4 + w_{e_1k_5}k_5 + w_{e_1k_6}k_6 \quad (2)$$

Similarly, for the evaluation of each emotion taking into account the information provided by the second mode (microphone) is done using formula 3.

$$em_{me_1} = w_{e_1m_1}m_1 + w_{e_1m_2}m_2 + w_{e_1m_3}m_3 + w_{e_1m_4}m_4 + w_{e_1m_5}m_5 + w_{e_1m_6}m_6 + w_{e_1m_7}m_7 \quad (3)$$

em_{ke_1} is the probability that an emotion has occurred based on the keyboard actions and em_{me_1} is the probability that refers to an emotional state using the users' input from the microphone em_{ke_1} and em_{me_1} take their values in $[0,1]$.

In formula 1 the k 's from $k1$ to $k6$ refer to the six attributes that correspond to the keyboard. In formula 2 the m 's from $m1$ to $m7$ refer to the seven attributes that correspond to the microphone. The w 's represent the weights of the attributes. These weights correspond to a specific emotion and to a specific input action and were calculated in the pre mentioned empirical study.

In cases where both modals (keyboard and microphone) indicate the same emotion then the probability that this emotion has occurred increases significantly. Otherwise, the mean of the values that have occurred by the evaluation of each emotion using formulae 1 and 2 is calculated.

The system compares the values from all the different emotions and selects the one with the highest value of the multi-attribute utility function. The emotion that maximises this function is selected as the user's emotion.

4.2 Agents That Act Emotionally

Through the incorporation of the OCC theory, the system may suggest that the educational agent should express a specific emotional state to the student for the purpose of motivating her/him while s/he learns. Accordingly, the agent becomes a more effective instructor.

In OCC theory, emotional states arise from cognitive models that measure positive and negative reactions of users to situations consisting of events, agents and objects. Correspondingly, events match user goals that are key elements in the OCC theory.

Tables 1 and 2 illustrate representative subsets of intensity variables concerning user input actions and application events that are used by the system's adapted OCC emotion model in order to propose an emotional state as a educational tactic for the animated agent. The variables illustrated in tables 1, 2 have been specified in our own implementation and adaptation of OCC in our educational application. The application's user interface is multi-modal, thus it is possible for the system to monitor and record user actions such as speed of typing through the keyboard as well as low voice volume through the microphone etc. The proposed authoring system integrates the OCC model by comprising a subset of five basic emotional states, namely happiness, sadness, anger, fear and surprise. Each one of the above mentioned five emotional states can be synthesized by the animated agent, as it is illustrated in figure 7.

Table 1. Variables for calculating the intensity of events for the OCC theory

Event variables
<ul style="list-style-type: none"> • a mistake (the user may receive an error message by the application or navigate wrongly) • many consecutive mistakes • absence of user action for a period of time • action unrelated to the main application • correct interaction • many consecutive correct answers (related to a specific test) • many consecutive wrong answers (related to a specific test) • user aborts an exercise • user aborts reading the whole theory • user requests help from the agent • user takes a difficult test • user takes an easy test • user takes a test concerning a new part of the theory • user takes a test from a well known part of the theory

Table 2. Variables for user actions through the microphone and the keyboard

Variables of user actions through keyboard and microphone
<ul style="list-style-type: none"> • user types normally • user types quickly (speed higher than the usual speed of the particular user) • user types slowly (speed lower than the usual speed of the particular user) • user uses the backspace key often • user hits unrelated keys on the keyboard • user does not use the keyboard • user speaks words from a specific list of words showing an emotion • user does not say anything • user speaks with a low voice volume (lower than the average recorded level) • user speaks in a normal voice volume • user speaks with a high voice volume (higher than the average recorded level)

As an example we describe a situation where a student is taking a multiple choice test after having read the corresponding theory of that lesson. The “default” goal for each user is to succeed in answering correctly the questions of each test. In our example we assume that the difficulty level of the test is high and the student has already answered a couple of questions correctly. At this point in accordance with the system’s incorporated OCC model the student is pleased that s/he has answered correctly the previous questions and is also experiencing hope that s/he will continue answering correctly. The corresponding intensity variables for this event are illustrated in table 1 as “many consecutive correct answers (related to a specific test)” and “user takes a test concerning a new part of the theory”. The second variable indicates that succeeding in such a test is difficult, thus can invoke admiration by the educational agent. The user’s hope of continuing to answer correctly may then be encouraged by the educational agent by expressing admiration for the student’s

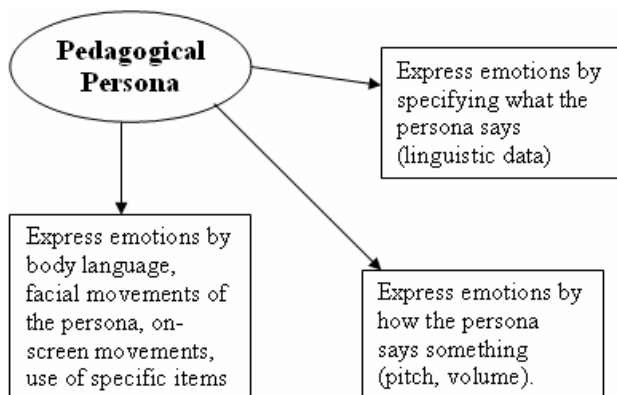


Fig. 7. Events-Actions of the agent for the synthesis of an emotional state

success and encouraging her/him to continue answering successfully. In this case the student has a “goal” for answering correctly. If the student continues her/his successful course the educational agent will express happiness, by saying something in a “happy voice” and/or by smiling or doing a positive gesture. This behaviour by the agent results by the analysis of the event variables of the interaction as well as by the goals both the student and the agent have set. The incorporation of the OCC theory provides the reasoning mechanism in deciding which emotional state is more appropriate for the agent in each sequence of events and user actions. Finally each one of the possible OCC states are translated by means of the five basic emotions the agent can express (for example confirmed hope, joy and admiration are OCC states that the agent expresses as happiness). Intensity variables as well as user and agents goals are illustrated in our simplified OCC model in figure 8.

5 Conclusions and Future Work

In this paper we have combined two well known theories, one from the field of decision making (SAW) and one from the field of cognitive psychology (OCC), in order to provide emotional interaction within an educational system. More specifically, we have described the implementation of an affective educational application that recognizes students’ emotions based on keyboard and microphone input actions and proposes tactics for the behaviour of educational agents based on pedagogical procedures. The resulting educational application employs a bi-modal user interface. For the elicitation of user emotions, as well as for providing a more user friendly environment, animated tutoring agents are present in each mode of the educational system. Their behavior is fully programmable and is controlled by the OCC model as a reasoning mechanism. All user input actions are processed through a decision making model which gives the system the important capability to make accurate hypothesis about the users’ emotional states.

It is among our future plans to evaluate the affective educational system in order to determine the degree of the system’s usefulness and usability for the students and also for their instructors. Furthermore, we intend to support the system’s multi-modal interaction by incorporating a third modality of interaction, visual this time [17]. By

incorporating the visual modality the system is expected to make even more accurate assumptions about the users' emotional state which will provide additional advantages to the educational process.

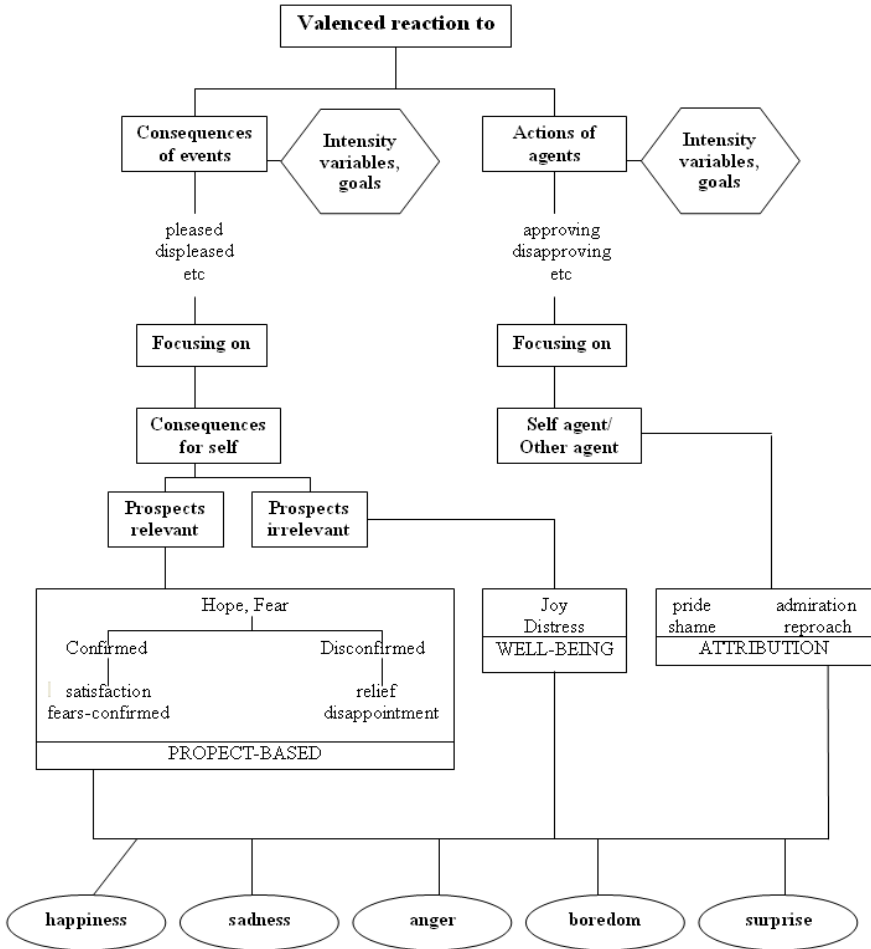


Fig. 8. Incorporation of the OCC model for specifying the agent's emotional state

References

1. Pantic, M., Rothkrantz, L.J.M.: Toward an affect-sensitive multimodal human-computer interaction. In: Proceedings of the IEEE, Institute of Electrical and Electronics Engineers, vol. 91, pp. 1370–1390 (2003)
2. Jascanu, N., Jascanu, V., Bumbaru, S.: Toward emotional e-commerce: The customer agent. In: Lovrek, I., Howlett, R.J., Jain, L.C. (eds.) KES 2008, Part I. LNCS (LNAI), vol. 5177, pp. 202–209. Springer, Heidelberg (2008)

3. Bernhardt, D., Robinson, P.: Interactive control of music using emotional body expressions. In: Proceedings on Conference on Human Factors in Computing Systems, pp. 3117–3122 (2008)
4. Lane, H.C. Intelligent Tutoring Systems: Prospects for Guided Practice and Efficient Learning. In: Whitepaper for the Army's Science of Learning Workshop, Hampton, VA (2006)
5. Brusilovsky, P.: Adaptive and Intelligent Technologies for Web-based Education. In: Rollinger, C., Peylo, C. (eds.) *Künstliche Intelligenz*, vol. 4, pp. 19–25 (1999); Special Issue on Intelligent Systems and Teleteaching
6. Johnson, W.L., Rickel, J., Lester, J.: Animated Educational Agents: Face-to-Face Interaction in Interactive Learning Environments. *International Journal of Artificial Intelligence in Education* 11, 47–78 (2000)
7. Lester, J., Converse, S., Kahler, S., Barlow, S., Stone, B., Bhogal, R.: The Persona Effect: affective impact of animated educational agents. In: Pemberton, S. (ed.) Proceedings of Conference Human Factors in Computing Systems, CHI 1997, pp. 359–366. ACM Press, New York (1997)
8. Virvou, M., Alepis, E.: Mobile educational features in authoring tools for personalised tutoring. *Computers & Education* 44(1), 53–68 (2005)
9. Leon, E., Clarke, G., Gallagher, V., Sepulveda, F.: A user-independent real-time emotion recognition system for software agents in domestic environments. *Engineering Applications of Artificial Intelligence* 20(3), 337–345 (2007)
10. Goleman, D.: *Emotional Intelligence*. Bantam Books, New York (1995)
11. Picard, R.W.: Affective Computing: Challenges. *Int. Journal of Human-Computer Studies* 59(1-2), 55–64 (2003)
12. Alepis, E., Virvou, M., Kabassi, K.: Knowledge Engineering for Affective Bi-modal Human-Computer Interaction. In: SIGMAP (2007)
13. Ortony, A., Clore, G., Collins, A.: *The cognitive structure of emotions*. Cambridge University Press, Cambridge (1990)
14. Alepis, E., Virvou, M.: Emotional Intelligence: Constructing user stereotypes for affective bi-modal interaction. In: Gabrys, B., Howlett, R.J., Jain, L.C. (eds.) KES 2006. LNCS (LNAI), vol. 4251, pp. 435–442. Springer, Heidelberg (2006)
15. Fishburn, P.C.: Additive Utilities with Incomplete Product Set: Applications to Priorities and Assignments. *Operations Research* (1967)
16. Hwang, C.L., Yoon, K.: *Multiple Attribute Decision Making: Methods and Applications*. Lecture Notes in Economics and Mathematical Systems, vol. 186. Springer, Berlin (1981)
17. Stathopoulou, I.O., Tsihrintzis, G.A.: Detection and Expression Classification System for Face Images (FADECS). In: IEEE Workshop on Signal Processing Systems, Athens, Greece (2005)

Author Index

- Alepis, Efthymios 277
Ameling, Michael 192
Anguiano, Eloy 218
Aranega, Vincent 137
Armendáriz-Íñigo, J.E. 205
- Beckhaus, Arne 85
Benfell, Adrian 18
Bergel, Alexandre 107
Berthold, Henrike 3
Bettini, Lorenzo 107
Braun, Iris 35
Bunse, Christian 247
- Callaway, Bob 46
Camacho, David 218
Cano, Juan I. 218
Cardoso, Jorge 3
Cuadrado, Félix 59
- Damiani, Ferruccio 124
Decker, Hendrik 233
Dekeyser, Jean-Luc 137
Dueñas, Juan C. 59
- Etien, Anne 137
- García-Carmona, Rodrigo 59
Giannini, Paola 124
González-Baixauli, Bruno 94
González de Mendivil, J.R. 205
Graf, Christian A. 85
Grottke, Michael 85
- Habib, Mursalin 46
Hattori, Satoshi 71
Herrera, Manuel 167
Höpfner, Hagen 247
- Izquierdo, Joaquín 167
- Juárez-Rodríguez, J.R. 205
- Kabassi, Katerina 277
Kaiya, Haruhiko 71
Karg, Lars M. 85
- Laguna, Miguel A. 94
Liroz-Gistau, M. 205
Liu, Kecheng 18
- Malgosa-Sanahuja, J. 181
Mansour, Essam 247
Manzanares-Lopez, P. 181
Montalvo, Idel 167
Mottu, Jean-Marie 137
Muñoz-Escóí, F.D. 205
Muñoz-Gea, J.P. 181
- Neumann, Dirk 85
- Patig, Susanne 150
Pérez-García, Rafael 167
Petersen, Rasmus Rosenqvist 263
Piñero-Escuer, P.J. 181
Pulido, Estrella 218
- Ricci, Alessandro 124
Rodríguez, Adolfo 46
Roychoudhury, Suman 247
Ruiz, José Luis 59
- Saeki, Motoshi 71
Sanchez-Aarnoutse, J.C. 181
Schill, Alexander 35, 192
Spillner, Josef 35
Springer, Thomas 192
- Viniotis, Yannis 46
Viroli, Mirko 124
Virvou, Maria 277
Voigt, Konrad 3
- Wiil, Uffe Kock 263
Winkler, Matthias 3
Wolf, Bernhard 192