Conferences in Research and Practice in Information Technology

Volume 143

Conceptual Modelling 2013

Australian Computer Science Communications, Volume 35, Number 9
Conceptual Modelling 2013

Proceedings of the Ninth Asia-Pacific Conference on Conceptual Modelling (APCCM 2013), Adelaide, Australia, 29 January – 1 February 2013

Flavio Ferrarotti and Georg Grossmann, Eds.

Volume 143 in the Conferences in Research and Practice in Information Technology Series. Published by the Australian Computer Society Inc.

Published in association with the ACM Digital Library.
# Table of Contents

Proceedings of the Ninth Asia-Pacific Conference on Conceptual Modelling (APCCM 2013), Adelaide, Australia, 29 January – 1 February 2013

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>vii</td>
</tr>
<tr>
<td>Programme Committee</td>
<td>viii</td>
</tr>
<tr>
<td>Organising Committee</td>
<td>ix</td>
</tr>
<tr>
<td>Welcome from the Organising Committee</td>
<td>x</td>
</tr>
<tr>
<td>CORE - Computing Research &amp; Education</td>
<td>xi</td>
</tr>
<tr>
<td>ACSW Conferences and the Australian Computer Science Communications</td>
<td>xii</td>
</tr>
<tr>
<td>ACSW and APCCM 2013 Sponsors</td>
<td>xiv</td>
</tr>
</tbody>
</table>

## Invited Paper

The Decision-Scope Approach to Specialization of Business Rules: Application in Business Process Modeling and Data Warehousing

*M. Schrefl, Bernd Neumayr, Markus Stumptner*

## Contributed Papers

When Grammars do not Suffice: Data and Content Integrity Constraints Verification in XML through a Conceptual Model

*Jakub Malý, Martin Nečasý*

Using Schematron as Schema Language in Conceptual Modeling for XML

*Soběslav Benda, Jakub Klímek, Martin Nečasý*

Letting Keys and Functional Dependencies out of the Bag

*Sebastian Link, Mozhgan Memari*

Improving Remote Collaborative Process Modelling using Embodiment in 3D Virtual Environments

*Erik Poppe, Ross Brown, Jan Recker, Daniel Johnson*

Using Formal Concept Analysis for Ontology Maintenance in Human Resource Recruitment

*Dominic Looser, Hui Ma, Klaus-Dieter Schewe*

Optimal Selection of Operationalizations for Non-Functional Requirements

*Amy Affleck, Aneesh Krishna, Narasimaha R Achuthan*

A Framework for Cost-Aware Process Management: Generation of Accurate and Timely Management Accounting Cost Reports

*M. T. Wynn, W. Z. Low, W. Nauta*

Automatic Data Transformation - Breaching the Walled Gardens of Social Network Platforms

*M. Wischenbart, Stefan Mitsch, Elisabeth Kapsammer, Angelika Kusel, Stephan Lechner, Birgit Pröll, Werner Retschitzegger, Johannes Schönböck, Wieland Schweinger, Manuel Wimmer*

## Author Index

*99*
Preface

This volume contains the proceedings of the 9th Asia-Pacific Conference on Conceptual Modelling (APCCM 2013), held at the University of South Australia in Adelaide, Australia from January 29 to February 1, 2013 as part of the Australasian Computer Science Week (ACSW 2013). The APCCM series focuses on disseminating the results of innovative research in conceptual modelling and related areas, and provides an annual forum for experts from all areas of computer science and information systems with a common interest in the subject. The scope of APCCM 2013 includes areas such as:

- Business, enterprise, process and services modelling;
- Concepts, concept theories and ontologies;
- Conceptual modelling and user participation;
- Conceptual modelling for decision support and expert systems; digital libraries; e-business, e-commerce and e-banking systems; health care systems; knowledge management systems; mobile information systems; user interfaces; and Web-based systems;
- Conceptual modelling of semi-structured data and XML;
- Conceptual modelling of spatial, temporal and biological data;
- Conceptual modelling quality;
- Conceptual models for cloud computing applications;
- Conceptual models for supporting requirement engineering;
- Conceptual models in management science;
- Design patterns and object-oriented design;
- Evolution and change in conceptual models;
- Implementations of information systems;
- Information and schema integration;
- Information customisation and user profiles;
- Information recognition and information modelling;
- Information retrieval, analysis, visualisation and prediction;
- Information systems design methodologies;
- Knowledge discovery, knowledge representation and knowledge management;
- Methods for developing, validating and communicating conceptual models;
- Models for the Semantic Web;
- Philosophical, mathematical and linguistic foundations of conceptual models;
- Reuse, reverse engineering and reengineering; and
- Software engineering and tools for information systems development.

The program committee has selected the contributed papers from 21 submissions. All submitted papers have been refereed by at least three international experts, and have been discussed thoroughly. The eight papers judged best by the program committee members have been accepted and included in this volume. The program committee invited Prof. Michael Schrefl from the Johannes Kepler University Linz, Austria to present the APCCM 2013 keynote. Prof. Schrefl is Head of the Data & Knowledge Engineering Institute and presented a talk on “The Decision-Scope Approach to Specialisation of Business Rules: Application in Business Process Modeling and Data Warehousing”.

The program committee selected the article When Grammars do not Suffice: Data and Content Integrity Constraints Verification in XML through a Conceptual Model by Jakub Malý and Martin Nečaský for the APCCM 2013 Best Paper Award. This includes a cash prize in the amount of AU$500, sponsored by the School of Information Management, The Victoria University of Wellington, New Zealand, and by the Advanced Computing Research Centre, University of South Australia, Adelaide. The article Using Schematron as Schema Language in Conceptual Modeling for XML by Jakub Klínek, Soběslav Benda and Martin Nečaský was selected for the APCCM 2013 Best Student Paper Award. This award includes a cash prize of AU$500, sponsored by CORE, Australia. Warmest congratulations to the authors of both papers.

We wish to thank all authors who submitted papers and all the conference participants for the fruitful discussions. We are grateful to the members of the program committee and the additional reviewers for their timely expertise in carefully reviewing the papers. We like to acknowledge the excellent work of the APCCM 2013 Publicity Chair Dr Markus Kirchberg and Dr Marko Baskovic.

Finally, we wish to express our appreciation to the local organisers at the University of South Australia for preparing the ACSW 2013 event in Adelaide.

Flavio Ferrarotti
Victoria University of Wellington, New Zealand
Georg Grossmann
University of South Australia
APCCM 2013 Program Chairs
January 2013
Conference Organisation

Program Committee Chairs

Flavio Ferrarotti, Victoria University of Wellington, New Zealand
Georg Grossmann, University of South Australia

Publicity Chair

Markus Kirchberg, National University of Singapore
Marko Boškovic, Research Studios Austria

Program Committee Members

João Paulo A. Almeida, Federal University of Espírito Santo, Brazil
Boualem Benatallah, University of New South Wales, Australia
Marko Boškovic, Research Studios Austria
Ross Brown, Queensland University of Technology, Australia
Denise De Vries, Flinders University, Australia
Gillian Dobbie, University of Auckland, New Zealand
Dragan Gašević, Athabasca University, Canada
Sven Hartmann, Clausthal University of Technology, Germany
Brian Henderson-Sellers, University of Technology Sydney, Australia
Markus Kirchberg, National University Singapore
Yasushi Kiyoki, Keio University, Japan
Ryan Ko, Hewlett-Packard Labs Singapore
Aneesh Krishna, Curtin University of Technology, Australia
Lam-Son Le, University of Wollongong, Australia
Chiang Lee, National Cheng-Kung University, Taiwan
Sebastian Link, University of Auckland, New Zealand
Jixue Liu, University of South Australia
Hui Ma, Victoria University Wellington, New Zealand
Christine Natschläger, Software Competence Centre Hagenberg, Austria
Shamkant Navathe, Georgia Institute Of Technology, USA
Martin Necas, Charles University, Czech Republic
Sudha Ram, University of Arizona, USA
Jan Recker, Queensland University of Technology, Australia Motoshi Saeki, Tokyo Institute of Technology, Japan
Michael Schrefl, Johannes Kepler University Linz, Austria
Nigel Stanger, University of Otago, New Zealand
Markus Stumptner, University of South Australia
Ernest Teniente, Universitat Politècnica de Catalunya, Spain
Riccardo Torlone, Roma Tre University, Italy
Qing Wang, Australian National University, Australia

Additional Reviewers

Mohsen Asadi, Simon Fraser University, Canada
Ralf Heese, Freie Universität Berlin, Germany
Michael Karlinger, Johannes Kepler University Linz, Austria
Bardia Mohabbati, Simon Fraser University, Canada
Organising Committee

Chair
Dr. Ivan Lee

Finance Chair
Dr. Wolfgang Mayer

Publication Chair
Dr. Raymond Choo

Local Arrangement Chair
Dr. Grant Wigley

Registration Chair
Dr. Jinhai Cai
Welcome from the Organising Committee

On behalf of the Organising Committee, it is our pleasure to welcome you to Adelaide and to the 2013 Australasian Computer Science Week (ACSW 2013). Adelaide is the capital city of South Australia, and it is one of the most liveable cities in the world. ACSW 2013 will be hosted in the City West Campus of University of South Australia (UniSA), which is situated at the north-west corner of the Adelaide city centre.

ACSW is the premier event for Computer Science researchers in Australasia. ACSW2013 consists of conferences covering a wide range of topics in Computer Science and related area, including:

- Australasian Computer Science Conference (ACSC) (Chaired by Bruce Thomas)
- Australasian Database Conference (ADC) (Chaired by Hua Wang and Rui Zhang)
- Australasian Computing Education Conference (ACE) (Chaired by Angela Carbone and Jacqueline Whalley)
- Australasian Information Security Conference (AISC) (Chaired by Clark Thomborson and Udaya Parampalli)
- Australasian User Interface Conference (AUIC) (Chaired by Ross T. Smith and Burkhard C. Wünsche)
- Computing: Australasian Theory Symposium (CATS) (Chaired by Tony Wirth)
- Australasian Symposium on Parallel and Distributed Computing (AusPDC) (Chaired by Bahman Javadi and Saurabh Kumar Garg)
- Australasian Workshop on Health Informatics and Knowledge Management (HIKM) (Chaired by Kathleen Gray and Andy Koronios)
- Asia-Pacific Conference on Conceptual Modelling (APCCM) (Chaired by Flavio Ferrarotti and Georg Grossmann)
- Australasian Web Conference (AWC2013) (Chaired by Helen Ashman, Michael Sheng and Andrew Trotman)

In additional to the technical program, we also put together social activities for further interactions among our participants. A welcome reception will be held at Rockford Hotel's Rooftop Pool area, to enjoy the fresh air and panoramic views of the cityscape during Adelaide’s dry summer season. The conference banquet will be held in Adelaide Convention Centre’s Panorama Suite, to experience an expansive view of Adelaide’s serene riverside parklands through the suite’s seamless floor to ceiling windows.

Organising a conference is an enormous amount of work even with many hands and a very smooth cooperation, and this year has been no exception. We would like to share with you our gratitude towards all members of the organising committee for their dedication to the success of ACSW2013. Working like one person for a common goal in the demanding task of ACSW organisation made us proud that we got involved in this effort. We also thank all conference co-chairs and reviewers, for putting together conference programs which is the heart of ACSW. Special thanks goes to Alex Potanin, who shared valuable experiences in organising ACSW and provided endless help as the steering committee chair. We’d also like to thank Elyse Perin from UniSA, for her true dedication and tireless work in conference registration and event organisation. Last, but not least, we would like to thank all speakers and attendees, and we look forward to several stimulating discussions.

We hope your stay here will be both rewarding and memorable.

Ivan Lee
School of Information Technology & Mathematical Sciences
ACSW2013 General Chair
January, 2013
CORE welcomes all delegates to ACSW2013 in Adelaide. CORE, the peak body representing academic computer science in Australia and New Zealand, is responsible for the annual ACSW series of meetings, which are a unique opportunity for our community to network and to discuss research and topics of mutual interest. The original component conferences - ACSC, ADC, and CATS, which formed the basis of ACSW in the mid 1990s - now share this week with eight other events - ACE, AISC, AUIC, AusPDC, HIKM, ACDC, APCCM and AWC which build on the diversity of the Australasian computing community.

In 2013, we have again chosen to feature a small number of keynote speakers from across the discipline: Riccardo Bellazzi (HIKM), and Divyakant Agrawal (ADC), Maki Sugimoto (AUIC), and Wen Gao. I thank them for their contributions to ACSW2013. I also thank invited speakers in some of the individual conferences, and the CORE award winner Michael Sheng (CORE Chris Wallace Award). The efforts of the conference chairs and their program committees have led to strong programs in all the conferences, thanks very much for all your efforts. Thanks are particularly due to Ivan Lee and his colleagues for organising what promises to be a strong event.

The past year has been turbulent for our disciplines. ERA2012 included conferences as we had pushed for, but as a peer review discipline. This turned out to be good for our disciplines, with many more Universities being assessed and an overall improvement in the visibility of research in our disciplines. The next step must be to improve our relative success rates in ARC grant schemes, the most likely hypothesis for our low rates of success is how harshly we assess each others’ proposals, a phenomenon which demonstrably occurs in the US NFS. As a US Head of Dept explained to me, “in CS we circle the wagons and shoot within”.

Beyond research issues, in 2013 CORE will also need to focus on education issues, including in Schools. The likelihood that the future will have less computers is small, yet where are the numbers of students we need? In the US there has been massive growth in undergraduate CS numbers of 25 to 40% in many places, which we should aim to replicate. ACSW will feature a joint CORE, ACDICT, NICTA and ACS discussion on ICT Skills, which will inform our future directions.

CORE’s existence is due to the support of the member departments in Australia and New Zealand, and I thank them for their ongoing contributions, in commitment and in financial support. Finally, I am grateful to all those who gave their time to CORE in 2012; in particular, I thank Alex Potanin, Alan Fekete, Aditya Ghose, Justin Zobel, John Grundy, and those of you who contribute to the discussions on the CORE mailing lists. There are three main lists: csprofs, cshods and members. You are all eligible for the members list if your department is a member. Please do sign up via http://lists.core.edu.au/mailman/listinfo - we try to keep the volume low but relevance high in the mailing lists.

I am standing down as President at this ACSW. I have enjoyed the role, and am pleased to have had some positive impact on ERA2012 during my time. Thank you all for the opportunity to represent you for the last 3 years.

Tom Gedeon
President, CORE
January, 2013
The Australasian Computer Science Week of conferences has been running in some form continuously since 1978. This makes it one of the longest running conferences in computer science. The proceedings of the week have been published as the *Australian Computer Science Communications* since 1979 (with the 1978 proceedings often referred to as *Volume 0*). Thus the sequence number of the Australasian Computer Science Conference is always one greater than the volume of the Communications. Below is a list of the conferences, their locations and hosts.

**2014. Volume 36. Host and Venue - AUT University, Auckland, New Zealand.**

**2013. Volume 35. Host and Venue - University of South Australia, Adelaide, SA.**

2012. Volume 34. Host and Venue - RMIT University, Melbourne, VIC.

2011. Volume 33. Host and Venue - Curtin University of Technology, Perth, WA.

2010. Volume 32. Host and Venue - Queensland University of Technology, Brisbane, QLD.


2008. Volume 30. Host and Venue - University of Wollongong, NSW.

2007. Volume 29. Host and Venue - University of Ballarat, VIC. First running of HDKM.

2006. Volume 28. Host and Venue - University of Tasmania, TAS.


1998. Volume 20. Hosts - University of Western Australia, Murdoch University, Edith Cowan University and Curtin University. Venue - Perth, WA.


1995. Volume 17. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Glenelg, SA.


1990. Volume 12. Host and Venue - Monash University, Melbourne, VIC. Joined by Database and Information Systems Conference which in 1992 became ADC (which stayed with ACSW) and ACIS (which now operates independently).

1989. Volume 11. Host and Venue - University of Wollongong, NSW.


1987. Volume 9. Host and Venue - Deakin University, VIC.

1986. Volume 8. Host and Venue - Australian National University, Canberra, ACT.


1983. Volume 5. Host and Venue - University of Sydney, NSW.

1982. Volume 4. Host and Venue - University of Western Australia, WA.

1981. Volume 3. Host and Venue - University of Queensland, QLD.

1980. Volume 2. Host and Venue - Australian National University, Canberra, ACT.

1979. Volume 1. Host and Venue - University of Tasmania, TAS.

1978. Volume 0. Host and Venue - University of New South Wales, NSW.
## Conference Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACDC</td>
<td>Australasian Computing Doctoral Consortium</td>
</tr>
<tr>
<td>ACE</td>
<td>Australasian Computer Education Conference</td>
</tr>
<tr>
<td>ACSC</td>
<td>Australasian Computer Science Conference</td>
</tr>
<tr>
<td>ACSW</td>
<td>Australasian Computer Science Week</td>
</tr>
<tr>
<td>ADC</td>
<td>Australasian Database Conference</td>
</tr>
<tr>
<td>AISIC</td>
<td>Australasian Information Security Conference</td>
</tr>
<tr>
<td>APCCM</td>
<td>Asia-Pacific Conference on Conceptual Modelling</td>
</tr>
<tr>
<td>AUIC</td>
<td>Australasian User Interface Conference</td>
</tr>
<tr>
<td>AusPDC</td>
<td>Australasian Symposium on Parallel and Distributed Computing (replaces AusGrid)</td>
</tr>
<tr>
<td>AWC</td>
<td>Australasian Web Conference</td>
</tr>
<tr>
<td>CATS</td>
<td>Computing: Australasian Theory Symposium</td>
</tr>
<tr>
<td>HIKM</td>
<td>Australasian Workshop on Health Informatics and Knowledge Management</td>
</tr>
</tbody>
</table>

Note that various name changes have occurred, which have been indicated in the Conference Acronyms sections in respective CRPIT volumes.
We wish to thank the following sponsors for their contribution towards this conference.

CORE - Computing Research and Education, www.core.edu.au

Australian Computer Society, www.acs.org.au

University of South Australia, www.unisa.edu.au/

School of Information Management, The Victoria University of Wellington www.sim.vuw.ac.nz

Advanced Computing Research Centre, University of South Australia acrc.unisa.edu.au
The Decision-Scope Approach to Specialization of Business Rules: Application in Business Process Modeling and Data Warehousing

Michael Schrefl\textsuperscript{1}  Bernd Neumayr\textsuperscript{1}  Markus Stumptner\textsuperscript{2}

\textsuperscript{1}Department of Business Informatics  
Data \& Knowledge Engineering  
Johannes Kepler Univ. of Linz - Austria  
Email: michael.schrefl@jku.at

\textsuperscript{2}Advanced Computing Research Centre  
Knowledge \& Software Eng. Lab  
University of South Australia  
Email: mst@cs.unisa.edu.au

Abstract

It is common in organizational contexts and in law to apply a decision-scope approach to decision making. Higher organization levels set a decision scope within which lower organization levels may operate. In case of conflict the regulations and rules of a higher level (e.g. European Union) take precedence over those of a lower level (e.g. a member state). This approach can also be beneficially applied to the specialization of the most important kind of business rules in information systems, action rules. Such rules define under which conditions certain actions may, must not, or need to be taken.

Applying the decision-scope approach to business process modeling based on BPMN means that business rules should not be buried in decision tasks but be made explicit at the flow level. This requires a re-thinking of the current BPMN modeling paradigm in that several aspects in conditional flow so far modeled jointly are separately captured: (a) potential ordering of tasks, (b) conditions under which a task may or may not be invoked, and (c) conditions under which a particular task needs to be invoked. Rather than re-defining BPMN as such, an appropriate extension may be provided on-top of BPMN and mapped to standard BPMN primitives.

Applying the decision scope approach to active data warehousing, where analysis rules express actionable knowledge, requires to consider two alternative hierarchies: (1) hierarchies of sets of points at the same granularity and (2) the roll-up hierarchy of points in multi-dimensional space.

This paper presents the decision scope approach, outlines how it complements inheritance and specialization approaches typically followed in object-oriented systems, and introduces consistency rules for business rule specialization as well as auto-correction rules, which rectify an inconsistent lower-level business rule such that it becomes consistent with higher-level business rules.

Keywords: Specialization, Inheritance, BPMN, Business Processes, Business Rules, Data Warehousing

1 Introduction

Specialization is one of the most prominent techniques in conceptual modeling to deal with complexity. It assists structuring design artifacts, making them more compact and better comprehensible. Artifacts are designed by specifying their features. With specialization, more specific artifacts inherit features from more general artifacts and may specialize inherited features through extension (i.e., adding orthogonal features) or refinement (i.e., adding details to inherited features); changing inherited features is termed “overriding”. Specialization is used in a number of related areas, such as knowledge representation [11], programming languages [9], data modeling [36], and business process modeling [12].

While the idea of organizing artifacts (nodes in semantic networks or concepts in ontologies, types in programming languages, classes of objects in data modeling, and classes of business cases in business process modeling) in hierarchies is common across related areas, the particular notions and rule governing specializations differ, sometimes even within the same area. A “rule-free” (i.e., arbitrary and unconstrained) change of inherited features, which is possible in some programming languages by free code overriding, is in discord with the very ideas of conceptual modeling and of abstracting common features along specialization hierarchies of design artifacts.

1.1 Contravariance and Covariance

It is generally agreed that propositions that hold for members of a superclass\textsuperscript{1} should also hold for members of a subclass, unless stated otherwise (in case of non-monotonic inheritance). The key differences between various approaches concern (1) what particular propositions over members of a class are made by a class specification and (2) the rules that govern specialization hierarchies (e.g., monotonic or non-monotonic inheritance).

In data modeling, classes usually specify constraints over properties of their members which are interpreted as invariant conditions that need to be satisfied by members of a class over their life time. In knowledge representation, classes (frequently called concepts) specify necessary and sufficient conditions for an object (frequently called individual) to be considered a member of a class; a change of an object’s property may lead to a dynamic re-classification.

In programming languages, the notion of type substitutability is predominant [14, 28]: Any member

\textsuperscript{1}Irrespective of the specific artifact, we will speak in the following of classes and objects, of members of classes, and of super- and subclasses for specialization relationships between classes.
(frequently called instance) of a class should be usable in any place in which a member of superclass is expected. Regarding the specialization with a class, type substitutability comes with the following proposition: “Given any member of the class, if the pre-condition of a method is met, the method can be successfully performed on that member”. Notice that the member of the class may also be a member of a subclass whose overridden method is not considered in this proposition. Type substitutability (in the presence of local static type checking) requires contra-variant inheritance - the redefined type of a parameter of an inherited method needs to be the same as or a supertype of the inherited type and a redefined precondition of a method must be more general (i.e., be implied by the inherited precondition). This ensures that if the precondition of a method is met at a superclass it is also met at the subclass.

Semantic data and process models have since their earliest appearance promoted the idea of co-variance as the more natural approach to specialization [10]; the “real world” argument for co-variant specialization [16] has been adopted also by UML 2.0 [13]. Regarding the methods of a class (or an activity of a process), co-variant specialization comes with the following proposition: “Given any member of the class, if the pre-condition of a method is met, the method cannot be successfully performed on that member”.

Notice the difference in the propositions implied by class specifications adhering to contra- or co-variant specialization, it is seldom expressed ad veribus.

Both notions, contra- and co-variance, have their place; usually for different purposes, substitutability and incremental specification. It has been suggested to combine both notions by selecting at run time a more general method for an instance of a more specific class if dynamic types of input parameters do not match the static types of input parameters of an overridden method [15]. Such approaches may have their merit in special cases, but lead to inconsistencies if specific constraints of a more specific class need to be met (which a more general method would not be aware of).

Specialization in business process modeling has been addressed by consistency notions that are akin to co-variance and contra-variance, such as observation vs. invocation consistency of object life cycles (used for modeling business processes in an object-centric manner) [34], or projection vs. protocol inheritance [5, 40] of process nets. Invocation consistency and protocol inheritance support the idea of substitutability in that a process class may be seamlessly replaced by another process class without any effect of the embedding environment. Observation and projection inheritance support the idea of incremental specification through extension and refinement.

1.2 Specialization and Business Rules

In re-active systems, such as active databases, the concept of specialization has been extended to active rules (that initiate an activity based on events and contexts it is common to apply a decision-scope approach to decision making. Higher organization levels define what is permitted and what is forbidden. Different to deontic logic, permissions and prohibitions are not complementary as they have a decision-scope in which lower organization levels may operate. Regulations and rules of a higher level (e.g., European Union) take precedence over those of a lower level (e.g. a member state) in case of conflict. Such an approach can also be beneficially applied to the specialization of business rules. We describe this in the
The decision-scope approach as described can be extended to activation conditions, which may trigger an activity if is permitted to do so. But not every activity that is permitted need to be actually invoked. There is a discretion to choose. This discretion is important in the context of conditional flow in business process modeling, where activation conditions may in effect chose any of several potentially applicable, i.e., permitted activities. In semi-formal decision making such a choice is deferred to the case officer. OMG promotes composition approaches in business process modeling, BPMN [35] where all flows and flow conditions are pre-defined and CMPM in which flow conditions and choices can be identified by a case manager on a case by case basis. The decision scope approach is applicable to both. For CMPM, it is required to consider different types of guards. For BPMN, multiple aspects in conditional flow so far modeled jointly need to be separately captured: (a) potential ordering of tasks, (b) conditions under which a task may or may not be invoked, and (c) conditions under which a particular task needs to be invoked. We will present such an extension to BPMN later in the paper and explain that rather than re-defining BPMN as such, an appropriate extension may be provided as well on top of BPMN and mapped to standard BPMN primitives (based on event-based gateways and deferred decisions).

Another dimension of choice in inheritance is monotonicity. Specialization and the decision-scope approach are inherently monotonic as any potential conflict of rules is resolved by precedence of rules of higher organizational levels over those of lower organizational levels. To avoid repeated specification of the same rules at lower organization levels, apart from a few exceptions, non-monotonic rules are used in law systems as well. These are explicitly identified as such by stating that a rule is presumed for lower jurisdictions and may be overridden by them (e.g., world wide sales conditions state that warranty is for one year, unless local consumer law prescribes a longer warranty period). Approaches in which monotonic and non-monotonic rules coexist in a decision scope approach have been employed in the context of authorization in database systems [7]: Authorization rules are classified into positive (corresponding to a permission) and negative (corresponding to a prohibition) rules and orthogonally thereto into strong (monotonic) and weak (non-monotonic) rules. Various rules govern the interplay of their combination in the context of inheritance along hierarchies of authorization objects, e.g., a weak positive authorization of a higher level may be canceled by a strong negative authorization rule at a lower level. This principle can be beneficially carried over to the specialization of business rules in general.

1.4 Decision Scope and Data Warehousing

The decision-scope approach is also relevant to active data warehousing [30, 38, 39], in which analysis rules express actionable knowledge, and to comparative data analysis in business intelligence [32], in which comparison results between two groups of facts (group of interest and group of comparison) may trigger actions providing guidance on how to proceed in analysis (guidance rules) or actions informing analysts about striking comparison results that may need further attention or action (judgment rules). We have applied some limited form of decision-scope approach in active data warehousing in the past and encountered the need to study specialization of analysis rules in the SemoCockpit project. In addition to business process modeling, two alternative hierarchies need to be considered: (1) hierarchies of sets of points at the same granularity and (2) the roll-up hierarchy of points in multi-dimensional space.

Specialization along subset relationships between sets of points at the same granularity is governed by the same rules as specialization between a superclass and a subclass of objects. Specialization along a roll-up hierarchy of points in multi-dimensional space actually concerns entities of different kinds, yet connected by some form of part-of relationship. This gives rise to two alternative rule evaluation strategies, both meaningful in practice, but with a different application space in mind.

In the prescriptive strategy, an action triggered for a higher level point implicitly implies the same action for every lower level point. E.g. if a company decides to abandon a product line (such as mobile phones) this decision is implied for every product (i.e., every phone model in our example) of this product line. In the presumed strategy, an action (typically an informative one) triggered for a higher level point is presumed to apply to lower level points and is thus not carried out again for lower level points (to avoid unnecessary information overload, the basis of performing roll-up analysis in data warehousing), unless it is justified by a negative activation rule. E.g., if in comparative data analysis a judgment rule triggers an informative action that average treatment costs for patients with diabetes mellitus of type 2 in Austria are twice as high than in Germany last year, it is presumed that this will hold in general for the province in Austria. Minor deviations are generally not of interest in comparative data analysis, but major ones are. The knowledge what kind of exceptions should be reported (a kind of business rule) can be expressed by a negative activation rule according to the notion of negative rules described before. Such a negative activation rule may trigger an informative action reporting a notable exception if the difference in compared costs is less than 20% for comparing patients in a province of Austria with patients in Germany but a positive activation rule has led to report at a higher granularity a striking difference of average treatment cost per patient in Austria versus Germany. E.g., based on such deviating observation for comparing Tyrol (a province of Austria) with Germany this rule will report that contrary to Austria as a whole, average treatment costs per patient have been similar when comparing Tyrol with Germany.

We discuss the decision scope approach to specialization of business rules from a general perspective in the remainder of this paper, which is organized as follows. In Section 2, we quickly revisit co- and contravariant specialization approaches in object-oriented systems and describe how the decision scope approach applied in organizational contexts reconciles both.
Section 3, we show by detailed examples how the decision-scope approach can be applied in the realm of business process modeling and propose an extension of BPMN to this purpose. We introduce a set of consistency rules that govern the decision scope approach and introduce a set of auto-correct-rules to modify inconsistent rules such that they become consistent. Continuing the law analogy, auto-correction amounts to repealing a simple law by a constitutional court if it contradicts constitutional law. In Section 4, we extend the decision-scope approach to data warehousing. Section 5 concludes the paper with a summary and outlook.

2 Inheritance: A Quick Review and an Introduction to the Decision Scope Approach

We quickly review the notions of co- and contravariant inheritance in conceptual modeling and object-oriented systems, introduce the decision scope approach to specialization as observed in law and organizational context, and show how co- and contravariant inheritance can be reconciled in the decision-scope approach by following both notions independently in parallel.

2.1 Inheritance in Conceptual Modeling and Object-Oriented Programming

The notion of specialization according to co-variance has been promoted by semantic data and process models as a natural approach to reflect how sets and subsets of objects of the real world relate. It supports information systems design by specializing a more general class through extension, i.e., by adding new properties (attributes or relationships), and refinement, i.e., by providing more detail for inherited properties or by restricting them. Usually, classes specify invariant conditions over properties of its members. These are treated as constraints that need to be satisfied by its members over their life time. Inherited invariant conditions may be strengthened at subclasses. Any member of a subclass is considered to be also a member of the superclass such that if inherited constraints (invariant conditions) are strengthened at subclasses a member of a subclass will also satisfy the more general constraints of its superclasses. In case of relationships between members of classes, a subclass of one side of the relationship may restrict the inherited relationship to be to a subclass of the original other side of the inherited relationship. Considering relationships does not add to the nature of the problem itself, we restrict our discussion to attributes of object classes with simple types (such as Integer, String or restrictions of these) from here on.

UML 2.0 [18] has also adopted co-variant specialization and we will use it to present our examples.

Example 1. We consider a very abbreviated description of selected parts of a loan application process in this paper. Figure 1 depicts a class hierarchy of loan applications in UML notation. Loans are described by an amount (a), a credit worthiness (w)\(^4\), and a monthly rate (r). Private loan applications are further described by the attachable income (i) that is of relevance for salary seizures should the private loan not be repaid. Mortgage loans are further described by a mortgage lending limit (l) of the mortgaged property that is of relevance if the mortgage is to be redeemed should the loan not be repaid.

\(^4\)Actually, the credit worthiness relative to the loan application is a function of available disposable income and requested loan amount; we abstract from such details here.

Invariants specify that amounts of loans are between 10 and 900k $, amounts of private loans between 10 and 700k $, and amounts of mortgage loans between 30 and 900k $.

<table>
<thead>
<tr>
<th>Class</th>
<th>Attributes</th>
<th>Invariants</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAN</td>
<td>amount: $</td>
<td>inv: amount (\geq 10) and (\leq 900)</td>
</tr>
<tr>
<td></td>
<td>creditWorthiness: Int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>monthlyRate: $</td>
<td></td>
</tr>
<tr>
<td></td>
<td>setAmount (a: $)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>precond: a (\geq 10) and a (\leq 900)</td>
<td></td>
</tr>
<tr>
<td>PRIVATE_LOAN</td>
<td>attachableIncome: $</td>
<td></td>
</tr>
<tr>
<td></td>
<td>setAmount (a: $)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>precond: a (\geq 10) and a (\leq 80)</td>
<td></td>
</tr>
<tr>
<td>MORTGAGE_LOAN</td>
<td>lendingLimit: $</td>
<td></td>
</tr>
<tr>
<td></td>
<td>setAmount (a: $)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>precond: a (\geq 30) and a (\leq 900)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Class Hierarchy of Loans

Strengthening of constraints over inherited object properties, either by restricting their type to a subtype (subclass) or by strengthening explicitly stated invariant conditions, require co-variant inheritance for “mutators”, i.e., methods that update the values of attributes. The domain of input parameters of inherited methods must be accordingly restricted to a subtype of the updated inherited property and pre-condition of inherited methods must be accordingly strengthened to reflect to stronger invariant condition at the subclass. This applies also to types of return values of methods and their postconditions. We repeat what we already mentioned in the introduction that a precondition of a method is to be interpreted as a proposition that “Given a member of the class, if the precondition of a method is not met, the methods cannot be performed on that member”.

Example 2. Method setAmount(a:$) of class LOAN, which sets the amount of a loan application to the value supplied as parameter, has a precondition coinciding with the corresponding invariant. Notice that in general preconditions of methods may be different to re-stating relevant invariants. In fact it should be unnecessary to include relevant invariant conditions in pre-conditions as these kinds of preconditions should ideally be inferred (at least for setter- and getter-methods). The precondition of method setAmount is overriden at subclasses PRIVATE_LOAN and MORTGAGE_LOAN to reflect the invariants of these classes accordingly. The specialization applied complies with co-variance as explained earlier.

Class hierarchies designed according to co-variance in conceptual data modeling match class hierarchies that would be built by ontology reasoners based on class descriptions that contain no reference to superclasses but contain explicit descriptions of otherwise inherited attributes.\(^5\) The main difference is that in conceptual modeling the class hierarchy is the primary design artifact along which attribute descriptions are inherited by subclasses from superclasses.

\(^5\)We assume here that concepts in the ontology refer to a common set of objects and that class descriptions are interpreted as concept expression stating that the concept comprises all objects of the given object class that possess the attributes of the class and satisfy the invariant conditions of the class.
whereas in current ontology engineering practice, the relationships of classes (concepts) to their attributes, but without reference to a superclass, are the primary design artifact and subclass relationships are inferred through subsumption reasoning.

There are - depending on the ontology and conceptual model used - other differences between ontologies and conceptual data modeling, but these are not relevant for the point we wish to make here that class hierarchies and corresponding concept hierarchies align one-to-one. While originally, ontologies were envisioned as an explicit formal specification of a conceptual model [19, 20], especially since the inception of Semantic Web research, ontology languages frequently take on the open world assumption, while conceptual modeling mostly adheres to the closed world assumption. Also, ontology languages treat conditions over attributes as conditions for classifying objects whereas data models typically treat conditions over attributes as constraints that may not be violated in updating a member of a class. But this is not a universal assumption, as since the early days of conceptual modeling primitive classes have been complemented by derived classes whose members are dynamically re-classified if they no longer meet the derivation condition of the derived class. When one takes into account the full history of conceptual modeling, many claimed differences between conceptual models and ontology languages become blurred, especially since within ontology research basic distinctions of approach such as the ability to infer a subsumption (inheritance) hierarchy by reasoning or to define it by specification depend on the language used (e.g., [4, 5]).

While conceptual modeling and ontology engineering would be expected\(^6\) to come up with generally the same hierarchy, but use different paths to achieve the same result, this “common” class hierarchy will usually not constitute a class hierarchy that satisfies type substitutability as demanded by the contra-variant approach to inheritance promoted in object-oriented programming. Contra-variant inheritance requires to generalize the types of input parameters of inherited methods to super types and to generalize inherited preconditions of methods in overriding. This approach ensures that any method that can successfully be applied on some instance of a class can also be successfully applied on any instance of subclass of the former. We repeat what we have already stated in the introduction: a precondition of a method is to be interpreted as proposition “Given a member of the class, if the precondition of the method is met, the method can be successfully performed on that member.”

Example 3. The class hierarchy of loans in Figure 1 violates substitutability. However, method safeSetAmount(a) of class LOAN with precondition \( a \leq 30 \) and \( a \geq 90 \) is “type-safe”. It can be applied on any member of class LOAN, be it a mortgage loan or a private loan.

2.2 The Decision-Scope Approach in Law: Constitutional Law before Simple Law

Law hierarchies and organizations use a decision-scope approach for decision making along hierarchies of law or organizational levels. Regulations and rules come (among others) in two typical forms, permission that allows actions and prohibitions that forbid actions. Permissions and prohibitions are not complementary. There are actions that are on the one hand not explicitly permitted and on the other hand not explicitly forbidden. This leaves room for maneuver in decision making at lower organizational levels. We also observe that legal cases usually are related to a most specific jurisdiction and in case of potential multiple jurisdictions there are tie-brakers to decide upon in which concrete jurisdiction a legal case is to be handled. Similarly, we assume the abstract super-class rule, which has been identified as good design practice, is applied in designing class hierarchies [23]. Each object belongs to a single, most-specific class that is concrete, i.e., has no subclasses.

The decision scope approach is governed by a simple, intuitive set of consistency rules:

1. Permission and prohibitions are always disjoint and they are complementary at a concrete bottom-level organizational entity / law.
2. What is permitted at a higher level is also permitted at lower levels.
3. What is forbidden at a higher level is also forbidden at lower levels.

If we take this perspective and revisit the notions of co- and contra-variant inheritance, we can observe that the propositions made by contra-variant specialization correspond to permissions and the propositions made by co-variant specialization stretch over permissions and the decision scope, or if negated, correspond to prohibitions (i.e., to what is forbidden).

This observation allows us to superimpose both kinds of propositions into a single, dual-faceted class description. Behavior attributed to the permissive part of the class description is type safe in the traditional sense of type substitutability. Behavior attributed to the prohibitive part of the class descriptions can be identified as “illegal”, i.e., such action cannot be performed on any member of the class. To decide whether behavior attributed to the decision scope is permitted or forbidden, one needs to consult lower level classes along the class hierarchy to the concrete class of an object. For concrete classes, the decision scope is closed, i.e. what is forbidden is not permitted and what is permitted is not forbidden. The dual facets of class descriptions of abstract classes become a single facet for concrete classes.

Example 4. The class hierarchy of loans in Figure 1 with methods setAmount and safeSetAmount of class LOAN already represents such a dual-faceted class description. However, an integrated description would only represent a single method with a positive (reflecting permission) and a negative (reflecting prohibition) precondition. Similarly, in an integrated description the current invariant shown for class LOAN would be converted into a negative invariant condition \( a < 10 \) or \( a > 900 \) (which if met when updating a member will constitute an invalid update, irrespective to which specific class the member belongs to) and complemented by a positive invariant condition \( a \geq 30 \) and \( a < 90 \) (which if satisfied when updating a member will constitute a valid update, irrespective to which specific class a member belongs to).

Figure 2 illustrates the decision scope approach and its relation to co- and contra-variant inheritance.

3 The Decision Scope Approach in Business Process Modeling

In this section we discuss how the decision scope approach can be applied to business process modeling.

\(^6\)Notice that we do not claim a one-to-one correspondence but consider the global picture.
In particular we consider the, in our opinion, most important kind of dynamic business rules: action rules. They concern under which conditions certain actions (activities, tasks) may, must not, or need to be taken in the process flow.

3.1 Business Rules in Business Process Modeling

Business rules in business process modeling express constraints over properties of business cases, the potential order of activities, pre- and postconditions of activities, and activation conditions (which express under which conditions a particular activity is invoked). At the design level, constraints over properties of business cases are typically captured by the modeling primitives of conceptual data models and association of the decision scope (e.g., by UML, cardinality constraints or OCL); the potential order of activities is typically captured by various forms of process diagrams based on Petri nets, state charts, or similar formalisms. Conditions under which activities may, must not, or need to be taken are frequently not considered independently from process flow. Applying a decision scope approach to modeling business rules that govern the execution of business processes requires to consider these kinds of conditions independently of each other and of process flow. But before we look into the specialization of business rules, we revisit the modeling of business processes from a general perspective.

The order of activities of a business process is typically expressed by some form of process diagram, e.g., by BPMN. Of particular interest are decision points at which process flow splits into alternatives.

Example 5. Figure 3 depicts a fragment of a loan application process. Once the credit worthiness for a loan application has been determined (not shown), the loan application is granted (g), rejected (r), or forwarded for decision to the board (f). Note: This is a simplified picture, the loan application process modeled may have other alternative paths at this decision point such as requesting additional information from the applicant if the case officer is not satisfied with the information at hand. Note: The loan application process modeled in full (in Petri-net notation) can be found in [26].

Activities are governed by business rules that define under which conditions an activity a of a process class O may be invoked \(P_a^O\), must not be invoked \(F_a^O\), or need to be invoked \(O_a^O\). To keep the explanation of the decision scope approach simple, we assume herein for simplicity that conditions are expressed over properties of a business case. However, the presented approach can be extended to conditions over parameters of activities and to events. Note that we use the terms permission \(P\), prohibition / forbidden \(F\), and obligation \(O\) from deontic logic; but different to deontic logic “permitted is not equivalent to not forbidden” and, conversely, “forbidden is not equivalent to not permitted”. Likewise, we will later use disobliged \(D\) as “counterpart” to obliged \(O\), where both conditions are again not complementary.

Example 6. Figure 4 visualizes the business rules governing activity forward depending on the amount of the loan and the credit worthiness of the applicant with respect to the loan application. (Note that different to Figure 1 we assume here and in the following that all kinds of loans are between 0 and 80k €. This is to simplify the presentation and to be able to focus on key points). A loan application with a low credit rating must not be forwarded to the board (as it should be immediately rejected). A loan application for a low amount and with a very high credit rating may (dependent on the discretion of the case officer) be forwarded to the board.

The white “open space” in Figure 4 constitutes the decision scope for activity forward for subclasses of class LOAN, which will be introduced later.

Note that we have used positive (+), negative (−), and activation \(\hat{a}\) conditions to specify permitted, forbidden, and obliged invocation of an activity. We intentionally use here a different terminology and notation \(\hat{a} \in \{+, -, \hat{a}\}\), for conditions indicated with activities as we will later auto-complete and auto-correct indicated conditions to permissions \(P\), prohibitions \(F\), and obligations \(O\).

Similarly, Figure 5 visualizes the business rules governing activity grant of class LOAN and Figure 6 the business rules for activity reject of class LOAN.

The business rules governing a single activity must be conflict-free (see: Figure 7), i.e., for the state of a business case (given by values of its properties): (1) an activity may not be permitted and forbidden as well, (2) an activity may not be obliged and disobliged as well, (3) if an activity is obliged it is permitted, and (4) if an activity is forbidden it is disobliged.

The business rules governing the set of alternative activities \(A\) at a given decision point in a business process must be decision-consistent (Figure 8), i.e., for a state \(\sigma\) of a business case at the decision point: (1) at least one activity must be permitted, and (2) at most one activity is obliged (which means it is permitted).
same time and it is possible that no activity is obliged for a given state of a particular instance of a business process (as determined by the values of its properties). Then it is at the discretion of a human process agent to decide which one of the permitted alternative activities to choose for a given process instance.

Considering the propositions we attached to permission and prohibition of an activity, i.e., that an activity may be executed for a business case if its permission is met and that an activity cannot be executed for a business case if its prohibition is met, we add two other consistency criteria such that these two propositions are correctly reflected in case of alternative activities: If an activity is obliged, alternative activities are forbidden (criterium no. 3 in Figure 8) and if an activity is permitted, alternative activities are disobliged (criterium no. 4 in Figure 8), which means they cannot be obliged at the class or some subclass. We note that criterium no. 2 is redundant. It follows from no. 4 in Figure 8 and no. 3 in Figure 7.

During business process design, adding an obligation to an activity may be restricted by the permissions defined with alternative activities and may require to extend the prohibitions specified with alternative activities such that criteria no. 3 and 4 of inter-activity consistency are met. To avoid such redundant specification, we introduce appropriate autocorrection rule in subsection 3.3.

Figure 4: Business process LOAN: Business Rules Regarding forward

Figure 5: Business process LOAN: Business Rules Regarding grant

Example 7. The business rules for activities forward (Figure 4), grant (Figure 5), and reject (Figure 6) of process class LOAN are intra-activity and inter-activity consistent.

Figure 9 shows the business rules for activities forward, grant, and reject for loans superimposed. Note that situations that can be inferred according to the consistency criteria are not specifically indicated: If an activity is obliged (shown) all other activities are forbidden (not shown), and if an activity is obliged (shown) it is permitted as well (not shown).

Business rules governing the permitted, forbidden, and obliged invocation of an activity in a business process should be explicitly represented at the design level rather than buried within activities. The latter approach would contravene the objective of conceptual design to make business rules explicit. Just as cardinality constraints, an important class of static business rules, are captured in UML graphically by

1. At least one activity permitted:
   \[ \exists a \in A : P_o^a(\sigma) \]
2. At most one activity is obliged:
   \[ a \in A, b \in A, a \neq b : O_o^a(\sigma) \rightarrow F_o^b(\sigma) \]
3. If an activity is enabled, alternative activities are forbidden:
   \[ a \in A, b \in A, a \neq b : O_o^a(\sigma) \rightarrow F_o^b(\sigma) \]
4. If an activity is permitted, alternative activities are disobliged:
   \[ a \in A, b \in A, a \neq b : P_o^a(\sigma) \rightarrow D_o^b(\sigma) \]
3.2 Specialization of Business Rules

Following the decision scope approach, action rules of a process class \( \hat{o} \) may be specialized at a subclass \( \hat{o} \) in that more specific rules at the subclass must act within the decision scope set by the superclass and providing room for manoeuvre at the subclass (see: Figure 11).

![Figure 11: The Decision-Scope Approach](image)

In particular, see Figure 12, (1) an action rule at the subclass must be more general than the same kind of rule at the superclass (generalization), (2) action rules at the subclass must not contradict those given at the superclass (super-class intra-activity conflict-freeness), and (3) (a) permissions and prohibitions for concrete classes are complementary, (b) an activity is obliged if all alternative activities are forbidden, and (c) an activity is disobliged if some other activity is permitted, and (d) obligations and dis-obligations are complementary (closed decision scope for concrete classes). Note that for checking overall consistency it would be sufficient to check first for generalization and then for activity-consistency (see above); the criterium “super-class intra-activity conflict-freeness” additionally captures a core principle of the decision-scope approach that superclasses should dominate subclasses. This will be of particular relevance when we later introduce auto-correction rules. We also note that one of the three criteria (3b-3d) is redundant.

We now apply this perspective to our use case.

Example 9. Figures 13 and 14 visually indicate the business rules governing activities forward, grant, and reject for private loans and mortgage loans respectively. As before with Figure 9, if an activity is shown obliged, all other activities are forbidden (not shown), and if an activity is shown obliged it is permitted as well (not shown). Two diagrams are shown for private loans, one covering the case where the attachable income \( i < r \) is less than the monthly rate \( r \), the other one where it is higher (in which case its is much riskier to grant a loan as rates cannot be fully recovered by salary seizes). Similarly, two diagrams are shown for mortgage loans, one covering the case where the amount \( a \) of the loan is at less than 70% of the mortgage lending limit \( l \) of the property, the other one where it is higher (in which case its is riskier to grant the loan as the loan amount may not be fully redeemable by sale of the property). Note: The application process for loans and the application process for private loans specialize the more general application process for loans by adding activities and states. They have been modeled in a Petri-net notation in [26] and are governed by the consistency criteria for process specialization presented in [34].

One can easily verify by visual inspection that the rules for mortgage loans and the rules for private loans
1. Generalization
   (a) \( P_\sigma^o(\sigma) \rightarrow P_\sigma^o(\sigma) \)
   (b) \( F_\sigma^o(\sigma) \rightarrow F_\sigma^o(\sigma) \)
   (c) \( O_\sigma^o(\sigma) \rightarrow O_\sigma^o(\sigma) \)
   (d) \( D_\sigma^o(\sigma) \rightarrow D_\sigma^o(\sigma) \)

2. Intra-Activity Conflict-Freeness
   (a) \( P_\sigma^o(\sigma) \rightarrow \neg F_\sigma^o(\sigma) \)
   (b) \( F_\sigma^o(\sigma) \rightarrow \neg D_\sigma^o(\sigma) \)
   (c) \( O_\sigma^o(\sigma) \rightarrow \neg D_\sigma^o(\sigma) \)
   (d) \( D_\sigma^o(\sigma) \rightarrow \neg O_\sigma^o(\sigma) \)

3. Closed decision scope for concrete class \( o \)
   (a) \( P_\sigma^o(\sigma) = \neg F_\sigma^o(\sigma) \)
   (b) \( O_\sigma^o(\sigma) = \bigwedge_{b \in A,b \neq o} P_\sigma^o(\sigma) \)
   (c) \( D_\sigma^o(\sigma) = \bigvee_{b \in A,b \neq o} P_\sigma^o(\sigma) \)
   (d) \( O_\sigma^o(\sigma) = \neg D_\sigma^o(\sigma) \)

Figure 12: Criteria for Decision Scope Consistency: Subclass \( o \), superclass \( \hat{o} \), state of business case \( \sigma \) are “decision-scope consistent” with those for loans in general (see: Figure 9). Moreover, note that no decision scope is left with mortgage loans and for private loans as regard to permissions and prohibitions. It is known for each activity whether it is permitted or forbidden.

Figures 15 and 16 show the representation of these business rules at the design level using the proposed extended BPMN notation. Notice that the diagrams shown provide a full picture for each activity, including inherited and redundant conditions (e.g., if an activity is obliged, the activity is permitted and alternative activities are forbidden). Auto-correction and auto-correction rules introduced below enable designers to specify much simpler diagrams (e.g., Figure 18).

3.3 Auto-Completion and Auto-Correction of Business Rules

Decision consistency can be ensured in several ways: (1) a priori by having reasoners checking for subsumption or disjointness of rule conditions according to the identified consistency criteria, (2) a priori by auto-correction and auto-completion of specified rule conditions in a way such that the consistency criteria are met, and (3) a posteriori by checking for conflicts during process execution. All three options are sensible and may also be used together, each for a different kind of consistency criterium.

A priori checking by reasoners can immediately identify decision conflicts but will usually limit the expressiveness of the language for writing rule conditions (a situation encountered frequently with ontology languages where languages supported efficiently by reasoners often do not meet the desired expressiveness). Moreover, it will require repeating rule conditions at subclasses, while in conceptual modeling subclasses usually only specify the “delta” (i.e., additional or changed features with respect to the superclass).

Auto-correction and auto-completion of specified activity conditions according to the intent of the decision-scope approach is more appropriate in general. Auto-correction and auto-completion rules determine based on given (but possibly incomplete and inconsistent) conditions, whether an activity is permitted, forbidden, or obliged for a given process state of a business case. Similar approaches are employed in database systems or programming languages. In database systems, the violation of an integrity constraint need not lead to a transaction abort but may be corrected in that other data are inserted, deleted, or modified as well such that all integrity constraints are finally met (e.g., the SQL-clause, ON DELETE CASCADE). In programming languages, pre-conditions of inherited methods are implicitly “or”-ed such that the criteria of contra-variant contract specialization are met (e.g., Eiffel).

Auto-correction and auto-completion take the role of remedying inconsistencies similar to the way in which simple law is repealed by a constitutional court if it is in conflict with constitutional law. The consistency criteria can be enforced as follows by auto-correction and auto-completion.

**Generalization**: Auto-completion in that an activity condition at the subclass is extended (using logical “or”) by the activity condition of the same kind at the superclass.

**Super-sub class intra-activity conflict-freeness**: Auto-correction in that the activity condition at the subclass is restricted (using logical “and”) to the decision scope set by the superclass.

**Closed decision scope for concrete classes**: There are two choices for auto-completion to close the decision scope for permission and prohibition, either permitting everything that is not forbidden or forbidding everything that is not permitted. Auto-completion for obligation is to set obligation of an activity to the
MORTGAGE LOAN

Figure 14: Business process MORTGAGE_LOAN. Business rules regarding grant, forward, and reject (superimposed)

conjunction of prohibitions of all alternative activities. And the decision scope between obligation and disobligation can be closed by setting disobligation to the complement of the obligation.

Intra-activity consistency: There are two choices for auto-correction of conflicts between permission and prohibition depending on whether prohibitions should take dominance over permissions or vice versa. Auto-correction is that the subordinate condition is restricted to the complement of the dominant condition, e.g., permission to negated prohibition. The same situation applies for obligation and disobligation. Inconsistencies between obligation and permission and between prohibition and disobligation can be rectified by auto-completion in that permission is extended by obligation and disobligation by prohibition.

Figure 15: Business process PRIVATE_LOAN in extended BPMN

...Figure 16: Business process MORTGAGE_LOAN in extended BPMN...
inter-activity consistency: Activity $a$ is forbidden for the case that an alternative activity is obliged and activity $a$ is disobliged for the case that an alternative activity is permitted.

The auto-correction rules fix several kinds of conflicts in one step: Interactivity conflicts between prohibition and permission (criterion no. 1 of Figure 7), whereby prohibition takes dominance; super-subclass intra-activity conflicts (criterion no. 2 of Figure 12), whereby according to the decision-scope approach conditions at the superclass are dominant; and inter-activity conflicts with regard to obligations, whereby obligation is restricted to cases in which no alternative activity is permitted or obliged. Note: The auto-correction rules are defined in a situation in which inter-activity conflicts between obligations and permissions have not been yet resolved (which will be done thereafter by auto-completion) such that the rules need to refer to prohibitions and positive activation-conditions.

The rationale behind the auto-completion rules is that designers can specify business rules in the same simple way as we presented them in Figures 9, 13, and 14. Thereby we assumed that (1) an activity is permitted if it is obliged, (2) an activity is forbidden if an alternative activity is obliged, and (3) an activity is disobliged if it is forbidden or if an alternative activity is permitted. Further, according to the decision-scope approach activity conditions at the subclass are extended by the corresponding activity conditions at the superclass (criterion no. 1 of Figure 12).

For classes without super classes, auto-completion and auto-correction is subject to the same rules whereby all conditions that refer to the superclass are to be substituted by “false”. For concrete classes, the decision scope for permissions and prohibition needs to be closed by extending for each activity its permission to the complement of its prohibition, its obligation to the conjunction of the prohibitions of alternative activities, and its disobligations to the complement of its obligation.

**Example 10.** Figure 18 shows business process MORTGAGE_LOAN, a concrete business process specializing business process LOAN of Figure 10, but incompletely and incorrectly defined. The introduced auto-completion and auto-correction rules produce the permissions, prohibitions, and obligations as expressed by the the activity conditions $+, -, \hat{\delta}$ shown for business process MORTGAGE_LOAN in Figure 16. Disobligations (not shown) are the complement of obligations.

![Figure 18: Business process MORTGAGE_LOAN, partially specified in extended BPMN](image)

**3.4 Discretionary Choice**

We have treated activation conditions of an activity as obligations in the previous subsection and reasoned that if an activity is obliged all alternative activities are forbidden. This is a meaningful assumption for modeling under which conditions an activity of a business process may, must not be, or needs to be invoked. In the case of service use, the situation is different. The user of a service may by discretion choose to select automatically (i.e., without human agent involvement) one of several permitted alternative activities. Such a discretionary choice trigger, however, does not imply that alternative activities in the used business process are forbidden in general. For service compatibility[17] it is only required that (1) choice implies permission, (2) a particular choice at a superclass implies the same choice at a subclass, and (3) choices are exclusive (in case of decision points between exclusive activities, which we consider herein), see: Figure 19. Discretionary choice conditions may be annotated to sequence flow in BPMN as another kind of condition.

![Figure 19: Criteria for Discretionary Choice](image)

**Example 11.** A discretionary choice condition may define that a loan application is automatically forwarded to the board if the loan amount is high and the credit rating is average $((w \geq 3 \land w < 5) \land a \geq 40)$,
which is permitted (cf. Figure 4). This choice condition may be extended for mortgage loan, e.g., to \((w \geq 3 \land w < 5) \land (a \geq 40) \lor (a \geq 60)\), which is permitted for mortgage loans (cf. Figure 14).

Note: The annotation of discretionary choice conditions is not shown in Figures 10 and 16.

Ways to auto-complete and auto-correct discretionary choice conditions are: (1) to restrict choice to permission, (2) to extend choice by the respective conditions of superclasses, and (3) to resolve conflicts in case multiple choice conditions overlap by (a) using priorities for activities, (b) by using a default master activity, or (c) by restricting each choice condition to non-overlapping parts.

3.5 Weak and Strong Business Rules

Specialization and the decision-scope approach as introduced are inherently monotonic as any potential conflict of rules is resolved by precedence of rules at a superclass over those at a subclass. In case the same rules apply for all subclasses apart from a few exceptions, monotonicity requires to restructure the class hierarchy or to redundantly define the same set of rules at several subclasses. Non-monotonic default rules may be a reasonable design alternative, providing such they are used in limited cases and are identified as such. In the legal context at which we looked for guidance to the decision-scope approach, lower level provisions may override higher-level provisions in identified cases (e.g., general world-wide sales conditions may state that warranty is for one year only, unless local consumer law prescribes otherwise). An approach for supporting monotonic and non-monotonic rules together has been developed in the context of authorization rules in database systems by introducing weak and strong authorization rules. We carry this idea over to the specialization of business rules in general.

Strong business rules are handled as described so far. Weak business rules may be overridden or canceled partially or fully at subclasses. In case of conflict between a strong and a weak business rule, the strong business rule takes precedence.

Figure 20 visualizes the same business use cases for strong and weak activation conditions. Figures 21 and 22 summarize the interplay between strong and weak activation conditions, negative and positive. The figures show how weak obligations and weak disobligations for invoking an activity \(a\) of class \(o\) without a superclass or with superclass \(\hat{o}\), resp., are derived from activation conditions (weak or strong, positive or negative) and have to be read in conjunction with Figure 17.

Example 13. Figure 23 shows a negative strong activation condition \((-\hat{a} f)\) for activity \(\text{forward}\) of class \(\text{LOAN}\) canceling partially the inherited positive weak activation condition for a rather low loan amount in case the detachable income \(i\) is lower than the monthly rate \(r\) and the loan amount is below 30k. It also visualizes the resulting weak obligation to invoke activity \(\text{forward}\) for a private loan (\(\hat{O}\)).

In semi-automatic decision making, an activity \(a\) will be automatically invoked for a business case of concrete class \(o\), if it is strongly obliged or if it is permitted and weakly obliged.

Example 12. Figure 20 visualizes the same business rules as Figure 9 but extended with a weak activation condition \((\hat{a} f)\) to forward a loan application to the board in case the credit worthiness is relatively high and the loan amount is not very low.

\[
\begin{align*}
\hat{O}_o^a &= \begin{cases}
\hat{a}^o \land \neg D_o^a & \text{if } \hat{a}^o \text{ is defined} \\
\text{false} & \text{otherwise}
\end{cases} \\
\hat{D}_o^a &= \begin{cases}
\neg \hat{a}^o \land \neg O_o^a & \text{if } \neg \hat{a}^o \text{ is defined} \\
\text{false} & \text{otherwise}
\end{cases}
\end{align*}
\]

Figure 21: Weak Activation Conditions: Cancelation, class \(o\) without superclass

Figure 22: Weak Activation Conditions: Overriding and Cancelation, subclass \(\sigma\), superclass \(\tilde{o}\)
4 Analysis Rules in Data Warehousing

In this section we describe how the decision-scope approach can be applied in data warehousing. Business rules in the context of data warehousing usually represent actionable knowledge on how to react on the presence or absence of (base or aggregated) facts in a data warehouse. We call such kinds of business rules analysis rules. They are used in active data warehousing for semi-automatic decision-making, providing a feedback loop to transactional database systems in that rule actions are transactions in transactional databases. Analysis rules are useful as well in comparative data analysis where they guide users during an analysis process, provide judgments or alert managers upon detection of interesting comparative situations (which are given through the comparison of a fact group of interest vs. a fact group of comparison). For simplicity, we introduce the decision-scope approach for non-comparative situations.

4.1 Data Warehouses and Classes of Facts

A data warehouse consist of a set of facts and a set of dimensions. Each dimension consists of a leveled hierarchy of roll-up nodes. For simplicity we assume a linear hierarchy to explain the decision-scope approach: (1) The roll-up hierarchy of nodes is a tree, (2) Each node belongs to a level, (3) All leaf nodes belong to the same level, (4) Any two nodes of the same level roll-up to nodes of a common level. The set of facts, usually referred to as cube, consists of multi-dimensional points, whose coordinates are leaf nodes of the dimensions, and a set of measure values. We refer to the levels of the coordinates of a multi-dimensional point as granularity.

Example 14. Figure 24 depicts the schema of a simple data warehouse with dimensions PRODUCT, LOCATION, and TIME and cube SALES&LOSSES with measures loss and sold. The figure shows also sample nodes of dimension PRODUCT: nodes milk and cream at level Product, nodes longLife and shortLife at level ProductCategory, and node food at level ProductLine.

Base facts (whose point coordinates are at the bottom granularity) roll-up to aggregated facts (or roll-up facts). In this fashion the measure of an aggregated fact is derived from the base facts that roll-up to the aggregated fact, i.e., whose coordinates directly or indirectly roll-up to coordinates of the aggregated fact. From here on, we simply speak of “a point (or fact) rolls-up to another point (or fact)” , when the point (or fact) directly or indirectly rolls-up to the other point (or fact). In top-down analysis, drill-down refers to the inverse of roll-up.

Example 15. Fact (Milk, Vienna, 28-11-2012, 5, 20) rolls up to fact (shortLife, Austria, 2012, 8, 27).

For applying the decision-scope approach to specialization of analysis rules, we consider a fact as object and the set of facts of a particular granularity (given by a level for each dimension) as a class. Furthermore, we can consider the class of facts of particular granularity \(G\) that roll-up to a given multi-dimensional point \(p\), denoted as \(p[G]\). In this notation the class of facts at a particular granularity \(G\) is the set of facts that roll-up to the root point \(r\), consisting of the all-node from each dimension, \(r[G]\). For brevity, a coordinate of a point or a level of a granularity is omitted if the coordinate is the root node of the dimension or the level is the root level, respectively. If a point \(p\) roll-ups to \(g\), then \(p[G]\) is a subclass of \(g[G]\). While class hierarchies with multiple inheritance can be constructed this way, we introduce the principles of the decision-scope approach in data warehousing for single inheritance. An extension to multiple inheritance requires to check the introduced consistency criteria for each superclass and it requires as well to apply the introduced auto-correction and auto-completion rules for each superclass.


4.2 Mono-granular Analysis Rules

We can apply the decision-scope approach as introduced for action rules defined over classes of business processes in the same way to analysis rules defined over classes of data warehouse facts if we consider only analysis rules that are defined over classes of facts of the same granularity. We call such analysis rules mono-granular.

Analysis rules are associated with a class of facts and determine, based on conditions over facts of the class, whether an action should be invoked or not. Thus, analysis rules specify a positive or negative activation condition, strong or weak. In most data warehouse settings initiated actions are always permitted.
such that negative and positive preconditions of actions need not to be modeled (but this could be done as well if needed the same way as introduced for business processes).

Analysis rules are usually checked for a subset of facts for which they are defined based on external or application events. E.g., an informative action may be initiated at the end of a temporal period (month or year) after evaluating facts of this period once they have been loaded into the data warehouse, or a guidance action (suggesting further analysis steps) may be initiated for facts that are part of a current analysis situation (consisting of a set of facts selected in the current analysis step of an analysis session [31]).

Specialization along subset relationships between points at the same granularity is governed by the same set of rules as specialization between superclasses and subclasses of objects.

**Example 17.** Figure 25 depicts the conditions under which a particular Product is to be de-listed ($\ddot{\text{d}}$) or not to be de-listed ($-\ddot{\text{d}}$) in a particular State dependent on the percentage of the loss-to-sold ratio ($l/s$). The analysis rules are specified for product line food and specialized for product categories shortLife and longLife.

![Figure 25: Mono-Granular Analysis Rules: Delist](image)

### 4.3 Multi-Granular Rules: Prerogative Evaluation

In addition to the subset relationship between set of points at the same granularity, the roll-up hierarchy of points in multi-dimensional space can be used to evaluate analysis rules for the same action along drill-down relationships (the inverse to roll-up). One possible evaluation strategy is the **prerogative evaluation strategy** in which an action triggered for a higher level point implicitly implies the same action for each drill-down point. We have applied this principle in similar form for active data warehousing in the past [38, 39]. An alternative evaluation strategy will be presented in the subsequent subsection.

In the prerogative evaluation strategy, analysis rules for the same action are evaluated top-down. We assume at first that for one granularity at most one analysis rule is defined by a condition pair, consisting of a positive and negative activation condition. If one of the two condition applies for a roll-up fact, analysis is completed, whereby the indicated action is triggered if the positive activation condition is satisfied. Only if both conditions are not satisfied, i.e., for the situation that a fact falls into the open space of the condition scope, the analysis rules for drill-down granularities and drill-down facts at these granularities are considered in further rule evaluation.

**Prerogative** evaluation of analysis rules along roll-up hierarchies in top-down manner can be combined with evaluation along class hierarchies of points at the same granularity as described in the previous subsection. At each granularity all analysis rules defined for classes of points at that granularity are considered, and for each fact the most specific class is chosen. For simplicity we assume here that the class hierarchy of points has been defined such that each fact falls into a single most-specific class. Once a decision has been determined for a fact at a given granularity, analysis stops; if no decision could be made due to an open decision scope, analysis continues with drill-down facts at a lower granularity for which an analysis rule is defined. Again, we assume herein for simplicity that the hierarchy of granularities for which analysis rules for a given action are defined is a tree. The approach for message-to-method binding introduced for multiple inheritance of cooperation contracts [33], which define methods that are polymorphic in multiple receivers, can be carried over in order to extend the presented approach for the case of multiple inheritance.

**Example 18.** Figure 26 defines analysis rules at the granularity of (Product,State) and at the lower granularity of (Product,City) for action delist(PRODUCT,LOCATION). Figure 27 shows sample facts and for each fact whether action delist is triggered for the fact in the multi-granular evaluation strategy.

The analysis rule for (shortLife) [Product,State] is the most specific rule for roll-up fact (milk,Austria,20) but does not apply, neither positively nor negatively. Therefore, the analysis rules for action delist are evaluated for drill-down facts and the analysis rule specified for (shortLife) [Product,City] triggers for fact (milk,Vienna,48), de-listing milk in Vienna.

Analysis rule (shortLife) [Product,State] applies to roll-up fact (cream,Austria,20) de-listing cream in Austria which implicitly implies that cream is no longer sold in every city in Austria. Therefore, analysis is completed and no longer checked for drill-down facts such as (cream,Vienna,15).

![Figure 26: Multi-Granular Analysis Rules: Delist](image)

### 4.4 Multi-Granular Rules: Presumed Evaluation

An alternative evaluation strategy to the prerogative strategy is the **presumed evaluation strategy**. In the presumed strategy an action (typically an informative one) is not triggered again for drill-down facts to avoid information load, unless it is justified by a negative activation condition.

**Example 19.** Figure 28 defines analysis rules at the granularity of (Product,State) and at the lower granularity of (Product,City) for action alertMg(PRODUCT,LOCATION). Figure 29 shows sample facts and for each fact lists whether action alertMg is triggered for the fact in the multi-granular evaluation strategy.
The analysis rule for \( \text{shortLife} \) \( [\text{Product}, \text{State}] \) is the most specific rule for roll-up fact \( \text{cream, Austria, 20} \) but does not apply, neither positively nor negatively. Therefore, the analysis rules for action delist are evaluated for drill-down facts and the analysis rule specified for \( \text{shortLife} \) \( [\text{Product}, \text{State}] \) triggers for fact \( \text{milk, Vienna, 48} \) alerting the manager for milk in Vienna.

The analysis rule \( \text{shortLife} \) \( [\text{Product}, \text{State}] \) applies to roll-up fact \( \text{cream, Austria, 20} \) alerting the manager for cream in Austria, with the alert presumed to include all drill-down facts. This avoids raising an alert for every city in Austria and shields the manager from information overload, since presumably if the aggregated fact has a strikingly unusual measure value, so will usually the drill-down facts from which the aggregated measure is computed, e.g. \( \text{cream, Salzburg, 52} \). But different to prerogative evaluation, an exception is reported if the alert should not have been raised due to a negative activation condition for a drill-down fact (e.g., for drill-down fact \( \text{cream, Vienna, 15} \)). An exception alert will be raised in such a case.

<table>
<thead>
<tr>
<th>class</th>
<th>1/s ≥ 40</th>
<th>1/s ≥ 15</th>
<th>1/s ≥ 40</th>
<th>1/s ≥ 45</th>
<th>1/s ≥ 45</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{food} ) [\text{Product}, \text{State}] \</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 20</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 20</td>
</tr>
<tr>
<td>( \text{longLife} ) [\text{Product}, \text{State}] \</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 20</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 20</td>
</tr>
<tr>
<td>( \text{shortLife} ) [\text{Product}, \text{State}] \</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 20</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 20</td>
</tr>
<tr>
<td>( \text{food} ) [\text{Product}, \text{City}] \</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 20</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 20</td>
</tr>
<tr>
<td>( \text{longLife} ) [\text{Product}, \text{City}] \</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 20</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 20</td>
</tr>
<tr>
<td>( \text{shortLife} ) [\text{Product}, \text{City}] \</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 20</td>
<td>l/s &lt; 10</td>
<td>l/s &lt; 20</td>
</tr>
</tbody>
</table>

Figure 28: Multi-Granular Analysis Rules: Alert-Manager

<table>
<thead>
<tr>
<th>point</th>
<th>1/s 1/4 alertMg</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{milk, Austria} )</td>
<td>20</td>
</tr>
<tr>
<td>( \text{cream, Austria} )</td>
<td>50</td>
</tr>
<tr>
<td>( \text{milk, Vienna} )</td>
<td>48</td>
</tr>
<tr>
<td>( \text{cream, Salzburg} )</td>
<td>52</td>
</tr>
<tr>
<td>( \text{cream, Vienna} )</td>
<td>15</td>
</tr>
</tbody>
</table>

Figure 29: Multi-Granular Presumed Evaluation on Sample Facts

5 Conclusion

In this paper we have outlined the general principles of the decision-scope approach to specialization of business rules. We have shown that the decisionscope approach used by law hierarchies and hierarchical organizations for decision making along hierarchies of laws or organizational levels can be beneficially applied to the specialization of business rules. We have presented a set of consistency criteria that govern the decision-scope approach and introduced auto-correction rules to adjust inconsistent business rules in that conflicts are resolved in favor of the superclass, like in the real world a higher-level authority overrules a lower-level.

In particular, we have exemplified how the decision-scope approach can be incorporated into business process modeling and how it can be used in capturing actionable knowledge in data warehousing. We currently work in the SemCockpit [32] project on employing the decision scope approach for specialization of guidance and judgment rules in the context of comparative data analysis in business intelligence.


CONTRIBUTED PAPERS
When Grammars do not Suffice: Data and Content Integrity Constraints Verification in XML through a Conceptual Model

Jakub Malý\textsuperscript{1}  
Martin Nečaský\textsuperscript{1}

\textsuperscript{1}Department of Software Engineering  
Charles University in Prague,  
Malostranske namesti 25, 180 00 Prague, Czech Republic,  
Email: \{maly, necasky\}@ksi.mff.cuni.cz

Abstract

Complex applications can benefit greatly from using conceptual models and Model Driven Architecture during development, deployment and runtime. XML applications are not different. In this paper, we examine the possibility of using Object Constraint Language (OCL) for expressing constraints over a conceptual model for XML data. We go through the different classes of OCL expression and show how each class can be translated into XPath constructs. Subsequently we show how the constraints can be checked using Schematron. We introduce a function library \texttt{OclX}, which provides constructs necessary to translate those OCL constructs that have no counterpart in XPath. With our tool, it is possible to check validity of OCL constraints in XML data.

Keywords: XML, OCL, Integrity Constraints, UML

1 Introduction

Development of a complex XML application involves tens or even hundreds of interconnected XML schemas and related queries and documents. In our previous work (Nečaský, Klímek, Malý & Mlýnková 2012), we proposed a conceptual model for XML application heavily utilizing UML ((Object Management Group 2007)) class diagrams for schema modelling. With our approach, it is possible to design XML schemas mapped to a conceptual model of the problem domain. This approach improves efficiency and is also less predisposed to errors, than creating the schemas separately.

Authors of UML models often find a need to add additional information to their diagrams to describe some properties of the system. Simple natural language comments can be ambiguous, thus a formal language OCL (Object Constraints Language, (Object Constraint Language Specification 2012)) has been developed for the purposes where more rigorous and dependable approach is required. Besides, formal OCL constraints can be automatically translated into executable code for a specific platform or implementation language. Such translation were developed e.g. for SQL or Java by the works of (Technische Universität Dresden 2012). The generated code can be used to check the validity of the constraints in the running system. In this way, OCL can become a very powerful tool in Model Driven Development (Miller & Mukerji 2003) scenarios, stepping in in those situations, where some significant system property can not be expressed using only the diagrams by themselves.

In our research, we focus on the XML platform. XML platform has its own toolset (so called XML stack) for solving analogous issues. Tree grammar languages (Murray et al. 2005) (XML Schema (W3C 2012)), Relax NG (Murata 2002), etc.) can be used to define structural properties of the XML data used in the system. Properties regarding the values and content of used data are checked by Schematron (ISO 2006) schemas. XSLT (W3C 2012) plays the role of the programming language and XPath (W3C 2011) is the ground expression language of the XML platform.

In this paper, we show how OCL can be used for modelling XML application and incorporated with our XML schema management approach. We propose a general algorithm for translating OCL into the expression language for XML - XPath. We extend our (structural) schema modelling framework and show how XML data can be validated using Schematron schemas generated from OCL constraints. With our approach, OCL constraint defined at the abstract layer can be reused for generating code for constraint verification in XML documents. It is no longer necessary to rewrite the constraint manually, which reduces both the costs of development and scope for errors. The rest of this paper is organized as follows. In Section 2, we formally introduce our conceptual model for XML data and show how integrity constraints can be defined. In Section 3, we describe translation from OCL expression to XPath expression and introduce our extensions of standard XPath. Section 4 presents our experimental implementation. In Section 5 we relate to the existing work and in Section 6 we conclude and indicate our future research.

2 Enriching Model with Constraints

In this section, we show how integrity constraints can be defined for XML schemas. In our approach, we model XML schemes using extended class diagrams at two levels, called platform-independent model (PIM) and platform-specific model (PSM).

2.1 Structural Model

We use UML class diagrams at the PIM level to create an abstract model of the system. In this paper, we focus primarily on the PSM level. At the PSM level, the user can define several PSM schemas. The purpose of the PSM level is to model the system using constructs closer to the selected platform and implementation technology (which, in our case, is XML). We use UML class diagrams at the PSM level as well, but with certain modifications. Each PSM schema models a set of XML documents and can be automatically translated into a schema in one of the structural (grammar based) XML schema languages (we
currently support XSD and Relax NG). For the rationale behind multi-layered modelling of XML and the details of our extension of UML for XML, we refer to (Nečaszký, Mlynářová, Klimek & Maly 2012). For the purposes of this paper, we will use the following definition of a PSM schema:

**Definition 1** A platform-specific (PSM) schema is a tuple \( S' = (S'_0, S'_1, C'_S, C'_E) \) of disjoint sets of classes, attributes, and associations, respectively, and one specific class \( C'_E \in S'_E \) called schema class.

- Class \( C'_C \in S'_C \) has a name assigned by function name, parent association assigned by partial function parentAssociation and a list of child associations assigned by function childAssociations.

- Attribute \( A' \in S'_A \) has a name, data type, cardinality and XML form assigned by functions name, type, card and xform, respectively. \( xform(A') = \{ e, a \} \). Attribute is associated with a class from \( S'_C \) by function class.

- Association \( R' \in S'_R \) is a pair \( R' = (E'_1, E'_2) \), where \( E'_1 \) and \( E'_2 \) are called association ends of \( R' \). \( R' \) has a name assigned by function name, name \( (R') \) may be undefined, denoted by name \( (R') = \lambda \). Both \( E'_1 \) and \( E'_2 \) have a cardinality assigned by function card and each is associated with a class from \( S'_C \) assigned by function participant.

A concrete PSM schema models a set of XML documents with certain structure defined by the schema. Table 1 shows how a specific XML format is modelled by a PSM schema.

An example of a PSM schema in Figure 1a models a schedule of matches in a tournament. The schema class MatchSchedule has a single child association named tournament, which specifies that the XML document must have a tournament root node. Names of associations model nesting of elements, attributes model XML attributes \( xform(A') = a \) — that is the case of regNo or elements with simple content \( xform(A') = e \). Figure 1b shows, what a document valid against the sample schema look like.

We mark PSM constructs with an apostrophe in order to distinguish them from PIM constructs. In this paper, we do not deal with PIM constructs, but we decided to keep the notation consistent with our other papers.

#### 2.2 Integrity Constraints

If we want to check integrity constraints that go beyond structure description, we need an additional instrument besides a diagram. UML specification provides Object Constraint Language (OCL, *(Object Constraint Language Specification 2012)*) for such purposes. OCL is an expression language over a UML model. Figure 1 shows several examples of integrity constraints for our sample schema.

OCL is a text based language, combining mathematical notation (used in e.g. first-order logic expressions) with principles known from functional languages. Its grammar allows for recursive building of formulas from subformulas (every formula can be represented by an expression tree).

The major part of this paper is devoted to the algorithm of translating OCL expression into XML Path Language (XPath) expressions, which will allow us to check OCL integrity constraints in XML data using Schematron. In the next section, we will enumerate the supported types of formulas defined in OCL specification and propose a translation to a corresponding expression in XPath.

#### 3 Validation of Integrity Constraints in XML Documents

In this section, we present an algorithm for automatic translation of PSM OCL scripts into Schematron schemas, which can be used to validate the integrity constraints in
XML documents. Schematron is a straightforward rule-based language. It consists of rule declarations, where every portion of a document matching a rule (match patterns follow the same syntax as in XSLT templates) is tested for assertions defined in that rule (assertions are expressed as XPath tests, assertion is violated, when the effective boolean value of the expression is false). The example rule in Figure 2 tests, whether every element Person has subelement Name.

Schematron schema, asserting over values, usually complement XSDs/RNG/DTD (grammar based languages), which prescribe the overall structure. It is possible to write constraints on structure in Schematron schemas as well, but structure description using the grammar based languages is usually easier to write and manage and is widely supported by the tools.

The recommendation of XML Schema 1.1 (W3C 2012a) allows to include some of the Schematron constructs directly in the XSD via assertions (assert) and report). However, there are additional limitations when using assertions in XSDs – only restricted XPath 2.0 tests are allowed, for example not all XPath axes are allowed. (the restriction are imposed in order to facilitate efficient processing of XSDs). The specification (W3C 2012a) describes in detail what restrictions are imposed on test.

The usage of XSLT patterns for contexts of rules and XPath expression for tests of asserts was chosen because those are technologies well established in the XML ecosystem. It also facilitates Schematron validation using an XSLT processor – an XSLT pipeline can be used to translate a Schematron schema S into a validation XSLT transformation T; T is executed upon the validated XML document and outputs structured information. Results produced by T are formatted using SVRL – Schematron Validation Report Language, which is part of Schematron specification (ISO 2006). The report contains the constraints that have been checked, which were violated and the locations of the errors.

The power of Schematron is thus determined by the power of XPath. As we will show later in this section, for some classes of OCL expressions, a corresponding construct in XPath does not exist. For such cases, we created a library of XSLT functions, called OclX. Functions from OclIX can be used in XPath expressions to provide sufficient expressive power. Since OclIX is implemented using pure XSLT, our approach does not require modification in Schematron validators – if the validator uses XSLT internally, it’s logic can be preserved providing that T; imports OclIX library (details of validation are described later in Section 4).

To start off, we show how the principal OCL constructs can be translated to a Schematron schema. This steps creates a skeleton of the schema. It is apparent that rules’ contexts and asserts in Schematron play the same role as contexts and invariants in OCL. Thus, by creating a rule for each OCL context declaration and adding an assert in the rule for each invariant, we can create a schema verifying the validity of PSM ICs. Table 2 outlines the rules for translation, Figure 3 shows a concrete example.

The crucial step of the algorithm, which will be explained in the rest of this section, is how to translate the PSM OCL invariant body expression O’ to an XPath expression X(O’). To achieve the desired property that the Schematron assert really verifies validity of the corresponding OCL expression the translation must follow the following principle:

**Principle 1 (consistency)** Let X(O’) be the XPath expression obtained by translating PSM OCL invariant O’. Then, the effective boolean value of X(O’) is true iff invariant O’ holds.

---

**Figure 2:** Different kinds of OCL expressions (source: OCL specification, Chapter 8.3)

**Figure 3:** Example of translation of principal OCL constructs
We will construct the expression so that their atomic value is always of XSD type xs:boolean. In that case effective boolean value equals to the value of the expression.

We will now look at the different kinds of OCL expressions, as they are depicted in Figure 2, and elaborate how they can be expressed equivalently in XPath. From now on, we will apply some restrictions on the class of considered OCL expressions. We will omit StateExp and MessageExp, since the notions of state and message (signal) have no counterparts in our domain (XML data). Due to space restrictions, we will also omit TypeExprs, which deal with casting, and we will also treat all collections as sequences. Due to the architecture of XPath data model, we will also not allow nested sequences in expressions. We will get back to the problem of nested sequences and different types of collections in Section 6. These conditions leave us with LiteralExp, IfExp, VariableExp, LetExp, two kinds of LoopExp (IteratorExp and IterateExp) and FeatureCallExp (which encompasses operations (and operators) and references to attributes and associations defined in the UML model). We also have to define consistent handling of variables.

In the rest of this article, we will delimit OCL expression in the text using guillemets, e.g. this: «x+y>1» is an OCL expression. We will use large uppercase letters for OCL literals and special literals «invalid» (representing erroneous value) and «null» (representing erroneous expression).

<table>
<thead>
<tr>
<th>OCL construct</th>
<th>Schematron construct</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCL script</td>
<td>Schematron schema</td>
</tr>
<tr>
<td>Constraint block</td>
<td>Pattern with a rule</td>
</tr>
<tr>
<td>Context classifier</td>
<td>Pattern id</td>
</tr>
<tr>
<td>Context variable</td>
<td>let instruction for a variable</td>
</tr>
<tr>
<td>Invariant</td>
<td>Assert</td>
</tr>
<tr>
<td>Invariant body</td>
<td>Expression in assert test</td>
</tr>
<tr>
<td>Error message</td>
<td>Failed assert text</td>
</tr>
<tr>
<td>Subexpression in error message</td>
<td>value-of instruction in assert</td>
</tr>
</tbody>
</table>

Table 2: Translation of principal OCL constructs

Variables

There are three ways a variable is defined in OCL. Each invariant has a context variable, which holds the validated object. It can be named explicitly (such as t in Figure 3) or, when no name is given, the name of the context variable is self. Iterator expressions (described later in this section) declare iteration variables (such as m in the expression «forAll(m | .)» in Figure 1c). Let expressions (described later in this section as well) define a local variable.

We will construct the expression in such a way that the following principle holds:

**Principle 2** Every OCL variable used in O’ corresponds to an XPath variable of the same name in XO. References to OCL variables (VariableExp) are translated as references to XPath variables.

The OCL context variable (with default name self or named explicitly) is common in all invariants declared for the context. Therefore, to declare corresponding variable, we can use Schematron sch:let instruction in each rule (line «<sch:let name="n" value="n"/>» in the example). Declaration of XPath variables for the other OCL variables (declared as a part of LetExp or LoopExp) will be created in accordance with Principle 2, as we will demonstrate later in this section.

LetExp Let expressions define a variable and initialize it with value. The variable can be referenced via VariableExp in the subexpression of the given LetExp. XPath 3.0 added a corresponding construct - let/return expression. Thus, the following principle is in accord with Principle 2.

**Principle 3** Let O’ be a LetExp expression «let x : Type = initExp’ in subExp’», where initExp’ and subExp’ are both OCL expressions and the latter is allowed to reference variable x. Then O’ is translated to an XPath expression X0:

\[
\text{let } \$x := X_{\text{initExp'}} \text{ return } X_{\text{subExp'}}
\]

LiteralExp OCL allows literals for the predefined types, collection literals (e.g. «Sequence(1,2,3)»), tuple literals and special literals «null» (representing missing value) and «invalid» (representing erroneous expression).

**Principle 4** OCL literals are translated according to the following table.

<table>
<thead>
<tr>
<th>OCL</th>
<th>XPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>predefined type literal</td>
<td>corresponding XSD primitive type literal</td>
</tr>
<tr>
<td>sequence literal, e.g. «Sequence(1,2,3)»</td>
<td>XPath sequence literal, e.g. «(1,2,3)»</td>
</tr>
<tr>
<td>tuple literal, e.g. «Tuple { name = 'John', age = 10 }»</td>
<td>XPath map literal (more about tuples in Section 3.4)</td>
</tr>
</tbody>
</table>

IfExp Conditional expression in OCL has the same semantics as in XPath, it can be translated directly.

**Principle 5** Let O’ be an IfExp expression «if cond' then thenExp' else elseExp'». Then O’ is translated to an XPath expression X0:

\[
\text{if } (X_{\text{cond}}') \text{ then } X_{\text{thenExp'}} \text{ else } X_{\text{elseExp'}}
\]

3.2 Translating Feature Calls

In the following subsections, we will show how we translate FeatureCallExp. There are two types of features in UML – properties and operations, which can be referenced via respective FeatureCallExprs, as depicted in a separate diagram in Figure 5. We will elaborate on both types – PropertyCallExp and OperationCallExp separately.

**PropertyCallExp** Examples of navigation expressions via PropertyCallExp are e.g. «p.player», «p.parent.parent.player» or «d.starts», «d.« in Figure 1. The first one navigates to an attribute start of class Tournament, the second one navigates an association end player which is a part of the association between classes Match and MatchPlayer. Every FeatureCallExp has a source (inherited from CallExp, see Figure 2). The source in the first example is a VariableExp «t», in the second example, the source is a VariableExp «m». The third example is a chain of three PropertyCallExprs. Two of the steps navigate PSM associations in upwards direction (distinguished by using parent instead of a name of the association). The source
in the third example is the expression «p.parent.parents», its source is «p.parent» whose source is «p». The whole navigation starts in class MatchPlayer, goes through classes Match and Tournament and ends in Player. Navigation to properties can be translated by appending an XPath step, which uses either child or attribute axis. Translation of navigation to association ends depends on the direction of the association and whether the association has a name or not (an association without a name means that the subtree under the association is not enclosed by a wrapping tag, thus no XPath navigation is added).

**Principle 6** PropertyCallExp is translated by appending an XPath step to the translation of the source expression. Let O’ be a PropertyCallExp expression «source.‘p’» and Xsource be a translation of subexpression source’. If ‘p’ navigates to an attribute A’ ∈ Sκ and n’ = name(A’), then O’ is translated to XO’ as follows:

\[
X_{O'} = \begin{cases} 
X_{O'}/child::n' & \text{if xform(A') = e} \\
X_{O'}/attribute::n' & \text{if xform(A') = a}
\end{cases}
\]

If R’ = (E’1, E’2) is an association and p’ navigates to one of its ends E’ ∈ {E’1, E’2}, O’ is translated as follows:

\[
X_{O'} = \begin{cases} 
X_{O'}/child::n' & \text{if name(R') = λ} \\
X_{O'}/attribute::n' & \text{if E' = E'2 ∧ name(R') = n'} \\
X_{O'}/parent::node() & \text{if E' = E'1}
\end{cases}
\]

**OperationCallExp** Application of predefined infix and prefix operators, calls of OCL standard library operations and calls of methods defined by the designer in the UML model all come under OperationCallExp. For a majority of predefined operators (such as «+», «and», etc.), a corresponding XPath operator exists as well. For those, where no corresponding XPath operator exists (e.g. «xor»), we added a corresponding function in OclX library (we do not include the exhaustive list in the paper, it can be found in the documentation for OclX). Similarly for the predefined operations. As for methods defined by the designer, currently, we do not consider modelling class methods at the PSM level. If they were to be added, calls would be translated to calls to user-provided functions.

**Principle 7** Every OperationCallExp O’ is translated into a call of corresponding operation/operator (with the same amount of parameters; the translation of the source expression in O’ becomes the first argument in XPath in XO’, followed by the translation of the operation arguments in O’). The corresponding operation/operator is either built-in XPath or defined in OclX library.

![Figure 5: Different kinds of FeatureCallExp expressions.](image)

### 3.3 Translating Iterator Expressions

Loop expressions, such as the following:

- \(source->exists(pz | pz.name = p.name)\),
- \(source->collect(d | d.match)\),

are archetypal for OCL – they perform the task of joins, quantification, maps and iterations. They are called using «<-» (same as all operations on collections), but instead of a list of parameters, the caller specifies the list of local variables and the body subexpression (see Figure 2).

There are several important facts regarding loop expressions:

1. There are two kinds of LoopExp, a general IterateExp and IteratorExp. The general syntax of IterateExp is:

   \(\text{iterate}(i : \text{Type} ; \text{acc} : \text{Type} = \text{accInit} | \text{body})\),

   where i is the iteration variable, acc accumulator variable, accInit the accumulator initialization expression and body is an expression, which can refer to variables i and acc. The result is obtained by calling body expression repeatedly for each member of the collection (which is assigned to i and acc is assigned the result of the previous iteration). The value of acc after the last call is the result of the operation. The general syntax of IteratorExp is:

   \(\text{iteratorName}(i : \text{Type} | \text{body})\),

   where iteratorName is one of the predefined OCL iterator expressions (such as exists, closure, etc.) or may be defined in user extension. i is the iteration variable and body is an expression, which can refer to the iteration variable i (and all other variables valid in the place where the iteration expression is used). The semantics of the IteratorExp depends on the concrete iterator. The semantics for the predefined operators is given in the specification.

2. Except closure, all other predefined iterator expressions (and a majority of collection operations) can be defined in terms of the fundamental iterator expression iterate, e.g. \(exists(it | body)\) is defined as \(\text{iterate}(it ; acc : \text{Boolean} = \text{false} | acc \text{ or body})\). Semantics of user-defined iterator expressions can be defined using iterate as well.

3. Iterator expressions forAll and exists (serving as quantifiers) together with boolean operators not and implies make OCL expressions at least as powerful as first order logic. Operation closure increases the expressive power with the possibility to compute transitive closures. Operation iterate allows to compute primitively recursive functions (for more on the expressive power, see (Mandel & Cengarle 1999)).

4. Multiple iteration variables, such as in

   \(\text{forAll}(v1, v2 | v1 < v2)\),

   are allowed for some expressions, but that is just a syntactic shortcut for nested calls:

   \(\text{forAll}(v1 | c->\text{forall}(v2 | v1 < v2))\).

5. Collection operations define variables (iterators and accumulator) are local (they are valid in the subexpression only). Other variables can be referenced from body expression as well, be it context variable (self) or variables defined by outer LetExp or LoopExp expressions. Variables except the iteration variables (and accumulator in iterate) are free in body expression.

For translation to XPath, property 2 implies that it is sufficient to show, how to translate closure and iterate, other operations can be defined using iterate. If we succeed, property 3 ensures that we can check constraints
with non-trivial expressive power, incl. transitive closures. Property 4 relieves us from the necessity to deal with expressions with multiple iterators. However, property 5 implies that we have to deal with local variables for iterator expressions.

There is no operation similar to iterate in XPath, nor can it be, in its most general form, expressed by some other XPath construction. However, we will show that iterate, and consequently all the other iterator expressions, can be implemented as XSLT higher-order functions.

Higher-order functions (HOFs) are a new addition proposed for the drafts of the common XPath/XQuery/XSLT 3.0 data model, which introduces a new kind of item – function item. With function items, it is possible to:

1. assign functions to variables, pass them as parameters and return them from functions,
2. function items can be called,
3. declare anonymous functions in expressions.

HOF is a function, which expects a function item as a parameter or returns a function item as a result. OCL loop expressions can be looked upon as HOF as well. We can expect a subexpression (body, see Figure 2), which is evaluated (called) repeatedly for each member of a collection. Property 5 mentioned above is important for the semantics – body subexpression can have free variables, which are, when evaluated, bound to variables defined in the source of the loop expression. E.g. in the expression IC3 in Figure 1c «t.match ->forall (m | m.start «t.start and m.end «t.end),» (the resulting collection) the first expression refers to two variables – m and t. Variable m is the iteration variable, variable t is free.

**Principle 8** IterateExp defines two variables, accumulator and the iteration variable. IterateExp defines one variable – the iteration variable. The translation of both IterateExp and IteratorExp must be in accord with Principle 2, i.e. these variables must be available as XPath variables in the translation of the body expression.

Figure 6a shows how iterate is implemented in OclX. It is a higher-order function, expecting a function item of two arguments in its third parameter body. The draft of XSLT 3.0 (W3C 2012c) introduces new instruction xsl:iterate, which we can use to our advantage. The function item is called repetitively for each member of the collection (line 10), with the two expected arguments – a member of the collection and the current value of the accumulator. When body was defined as an anonymous function item, the free variables it contains are bound to the variables available in the calling expression, which is in accord with the semantics of loop expressions of OCL. The second part of Figure 6a shows how HOFS exists can be defined in terms of HOF iterate. The definition utilizes an anonymous function node (line 23), which calls the function item passed as argument.

**Principle 9** Every IterateExp (call of iterate) is translated as a call of OclX HOF iterate. Every IterateExp (call of some iterator expression, such as exists etc.) is translated as a call of an OclX HOF of the same name. OclX contains a HOF definition for each predefined iterator expression. Subexpression body is translated separately and the resulting $X_{body}$ is passed as an anonymous function item to the HOF call.

3Some iterator expression can be in some cases translated using native XPath constructs without the need to call a HOF, e.g. exists can be translated using some/satisfies expression. Due to the space limitations, we do not discuss this sort of rewriting of the queries in this paper. Our experimental implementation (Klímek et al. 2012) allows the user to choose where several translations are possible.

To conclude the part about iterator expressions, we will address operation close. The syntax for close is the same as for other iterator expression, but the difference is that the semantics of close is not defined in terms of iterate – whereas the amount of iterations needed to compute iterate is fixed, close computes a transitive closure of the body subexpression (the resulting collection must be in depth first preorder) – thus, it is not known, how many calls of body will be required. Again, there is no equivalent construct in standard XPath. The implementation of close in OclX is depicted in Figure 6b.

### 3.4 Tuples

In this subsection, we show, how OCL expressions using tuples (anonymous types) can be translated to XPath. OCL allows the modeller to combine values in expressions into tuples. Tuples have a finite number of named parts and are created using TupleLiteralExp, a specialization of LiteralExp. An example of a tuple literal may be «Tuple {firstName = 'Jakub', lastName = 'Maly', age = 26 »». The values of the parts may be of arbitrary type, including collections and other tuples. The names of tuple parts (firstName, lastName, age in the example) must be unique and are used to access the parts of the tuple in expressions, similarly to attributes of classes (using "." notation), i.e. it is possible to write e.g. «employees->collect(e | Tuple {name = e.name, salary = e.salary })->select(l | l.salary > 2000)». The result of this expression would be a collection of tuples. Tuples are also closely related to operation product, which computes a Cartesian product of two collections: product(c1:Collection(T1), c2:Collection(T2)) = 

```
self->iterate(e1 | e1.acc = Set{} | c2->iterate (e2 | acc2 = acc1 + acc2 | including(Tuple[first = e1, second = e2]))
```

The result of product is a collection of type «Collection(Tuple(first : T1, second : T2))», which contains all possible pairs where the first compound comes from collection c1 and the second collection c2. This operation thus finalizes the suite of equivalents of the constructs required for a language to be relationally complete (see (Codd 1972)):

1. Select - can be expressed using select iterator expression,
2. Project - can be expressed using collect iterator expression that creates a tuple with the projected attributes (see the employees example above, which, in fact, performs projection to attributes name and salary),
3. Union - OCL has union operation as well,
4. Set difference - OCL has operation '-' working on sets,
5. Cartesian product - can be expressed using product,
6. (Rename) - can be expressed using collect in the same manner as project operation.

Thus, not only tuples can be used to write more concise expressions, but, together with the operation product, they increase the expressive power of the language to relational completeness (see (Mandel & Cengarle 1999)) for more on expressive power of OCL. Not supporting tuples would reduce the expressive power, thus we will elaborate on the possibilities of expressing them in XPath in this subsection.

Our proposed approach is to use map items. Map item is an additional kind of XPath item, which was added in the Working Draft of XSLT 3.0 (W3C 2012c)
(a) Functions iterate and exists

(b) Function closure

Proceedings of the Ninth Asia-Pacific Conference on Conceptual Modelling (APCCM 2013), Adelaide, Australia

(3.5 Error Recovery

OCL as a language has a direct approach to "run-time" errors or exceptions. Errors in computation cause the result of the expression to be invalid – a special value, sole instance of type OclInvalid. It conforms to all other types (i.e. it can be assigned to any variable and can be a result of any expression) and any further computation with invalid results in invalid – except for operation oclIsInvalid, which returns true, when the computations results in invalid and false otherwise. This operation thus provides the only, very coarse-grained error checking (there are no error codes or exception types) available in OCL. Unlike OCL computation, XPath/XSLT 2.0 processor halts when it encounters a dynamic error and there is no equivalent of oclIsInvalid. It is also not possible to instruct it to jump to the validation of the next IC when a computation of one expression fails.

XSLT 3.0, however, introduces new instructions – xsl:try and xsl:catch – which provide means of recovery from run-time errors. With these instructions, it is possible to implement oclIsInvalid as depicted in Figure 9. We, again, utilize higher-order functions capabilities – the expression is evaluated in a function call wrapped in try/catch. OCL expression oclIsInvalid(1/0) can be translated to oclX:oclIsInvalid(function() { 1 div 0 }) . OCL also allows invalid literal to be used explicitly in expressions (to indicate error). We translate this literal to a call to OclX function invalid(), which simply throws a dynamic error.

Optionally, our validation pipeline (fully introduced in section 4) allows to safe-guard the evaluation of each expression using try/catch, so that the validation of another constraint may continue if a runtime error occurs and it is
not contained by oclIsInvalid. In debug mode, detailed info is given using xsl:message.

```xml
<xsl:if test=""\$debug""><
    <xsl:message select=""Runtime error making the result invalid. ""/></xsl:message>
</xsl:if>

Figure 9: Implementation of oclIsInvalid using xsl:try/xsl:catch

**Principle 11** Calls of functions oclIsInvalid and oclIsUndefined are translated into calls of corresponding OclX HOFs, implemented using try/catch instructions. Usages of invalid literal are translated into calls of invalid().

### 4 Implementation

We incorporated the algorithm presented in this paper into our experimental XML schema management tool eXolution (Klímek et al. 2012). The tool allows for PIM schema modelling, semi-automatic PSM schema derivation. The user can specify integrity constraints at both PIM and PSM level. The user can also ask the tool to suggest which PIM constraints are relevant for a selected PSM schema. The tool examines the mapping between PIM and PSM levels and chooses which PIM constraints are applicable and tries to perform automatic translation. We do not describe this algorithm in this paper, but we want to emphasize that the tool allows to reuse integrity constraints from the PIM level and it is not necessary to create the same constraints at the PSM level manually.

The schema outputed by the tool can not (generally) be used by a standard validator, because it may contain references to OclX XSLT functions, which XPath does not know. Thus, we provide a modified Schematron pipeline (the pipeline can be run either using XProc (W3C 2010) or as a shell script) and requires an XSLT 3.0 Working Draft compliant processor (we tested our implementation using Saxonia 2012). The workflow is depicted in Figure 10.

The schema of usage of OCL for XML validation is depicted in Figure 10. When the user specifies ICs at the PIM level (1), the tool can helps him to transfer them to the PSM level – the tool selects relevant schemas and offers automatic translation where possible (2). Apart from ICs transferred from the PIM level, it is possible to create expressions solely for the PSM level (3). PSM OCL script can be generated (4), two variants are offered – for schema aware XSLT processors and non-schema aware (in that case, typed values are created from string values using constructors). Since some expressions may refer to OclX functions, OclX library must be linked (5) to the validation stylesheet, which are generated by Schematron pipeline transformations. To achieve this, we slightly modified Schematron pipeline transformations so that they add necessary imports. Schematron pipeline outputs a validation stylesheet (6), which can be used to validate XML data (7) – the result of validation is a Schematron Validation Report Language document (8), which contains a report on which constraints were checked, whether some of them were violated and if so, the locations of the errors. In this paper, we focused primarily on step (4).

In our paper (Malý & Nečaský 2012), we elaborate more on the implementation in XSLT, where we also on the differences between XSLT 2.0 and XSLT 3.0 with respect to corresponding OCL constructs.

To conclude this section, we show, how the example integrity constraints from Figure 1c and Figure 8 are translated. The resulting Schematron schema is depicted in Figure 11. Translation of constraint IC1 is straightforward. Constraint IC2 contains a call of a higher order function forAll. Anonymous function item is created for the body function and the body function also references a free variable i which is the context variable of the expression. Constraint IC3 illustrates the translation of nested iterator expressions (forAll and exists) into higher-order functions and also upwards association navigation (using parent XPath axis). Constraint IC4 uses oclIsUndefined (which tests for null value).

The last constraint IC5 is a translation of the OCL invariant from Figure 8 and demonstrates let expressions (definition of a local variable), iterate operation and tuples.

### 5 Related Work

Existing academic work (Wenfei & Jerome 2003), (Arenas et al. 2008) in the area of integrity constraints for XML focuses mainly on the fundamental integrity constraints known from relational databases – keys, unique constraints, foreign keys and inverse constraints – and their mathematical properties, such as decidability, consistency, tractability (with separate results for one-attribute vs. multi-attribute and relative vs. absolute keys). In this paper, we deal with general constraints in the form of arbitrary expressions.

Several approaches for modelling XML using UML were proposed (Conrad et al. 2000, Dominguez et al. 2011, Routledge et al. 2002), but they deal mainly with modelling the structure of the schemas, without debating the integrity constraints present in the model.

OCL and UML and related technologies are being researched (Hussmann et al. 2000) at Technische Universität Dresden, which is also the coordinator of the leading open-source implementation – Dresden OCL (Technische Universität Dresden 2012). Dresden OCL research was mainly targeting relational databases platform (Demuth & Hussmann 1999). A generic framework for generating translation OCL expressions into other expression languages was proposed in (Heidenreich et al. 2008). It mentions 2 applications: OCL → SQL translation and also OCL → XQuery (Boag et al. 2007). The expression are translated into the target language via patterns. It expects much tighter mapping between UML model and XML schema (unlike PIM/PSM schemas used in our approach, it does not consider regular properties of schemas). The OCL → SQL patterns are based on (Demuth & Hussmann 1999), OCL → XQuery on (Gaafar & Hussmann 1999).
The authors support constructs corresponding to projection, Cartesian product and restriction in the expressions (omitting the general iteration and closures facilities).

Authors of (Gaafar & Sakr 2004) examine the fundamental similarities of the two expression languages – OCL and XQuery. They propose a mapping from XQuery queries to OCL constraints (bottom-up approach). They show how the parts of elementary XQuery expressions can be mapped to OCL constructs, but they do not elaborate on translating definitions of and references to (local) variables, which would be interesting for queries with multiple variables (such queries correspond to more complex OCL iterator expressions, which are not mentioned in the paper). In consequence, the full expressive power of OCL is not harnessed (for more on expressive power of OCL, see (Mandel & Cengarle 1999)). In our paper (Malý & Nečasý 2012), we describe, how OCL iterator expressions can be translated to higher-order functions (new additions to the common XPath/XQuery/XSLT 3.0 data model working draft (W3C 2012b)).

6 Conclusion

Our aim in this paper was to further utilise the potential of MDA in XML applications by allowing the reuse of integrity constraints defined at the platform independent level in a UML diagram. We presented an algorithm for translation of OCL expressions into XPath expressions. We have shown one application of this translation algorithm – document validation. With our approach, it is possible to automatically generate integrity constraint checking code in the form of a Schematron schema, which can be used to validate XML document. We identified several classes of expressions, where standard Schematron is not sufficient, and proposed extensions required to preserve the expressive power of OCL. We incorporated the approach into our schema management tool exolutio (Klímek et al. 2012). Since our extension has a form of an XSLT function library, Schematron schema generated by our tool can be processed by any XSLT 3.0 compliant processor using modified Schematron pipeline.

In our future work, we plan to further improve the OCL to XPath mapping and add support for nested collections and collections of other kinds besides sequences, i.e. sets, bags and ordered sets. These types are alien to XPath data model, which only knows flat sequences. XSLT 3.0 Working Draft proposes an additional type to the XPath type system – a map (which we have already utilised for representing tuples in Section 3.4). A distinct feature of XSLT maps is that it allows both maps and sequences as values, thus, using maps, it is theoretically possible to represent nested collections (the syntax of some expressions however becomes a bit convoluted). We intend to improve the algorithm by proposing a coherent way for representing nested collections of any of the four kinds together with a syntax that is as transparent as possible.

In some cases, an expression can be translated in a more succinct way, i.e. a call of select OCL iterator expression can be translated using an XPath function (collection[condition]) rather than via a OcIX HOF call, as determined by Principle 9 in Section 3. Our implementation already offers a choice where several options of translation can be found. In our future work, we want to formalize the possibilities of rewriting expressions (and preconditions for each rewriting).

We also want to follow up our research in the area of document adaptation (Malý et al. 2011), where we proposed an algorithm for generating an adaptation script to transform documents valid against one version of a schema into documents valid against other version of the same schema. There are scenarios, in which document adaptation can utilise integrity constraints to precisely specify the mapping between the two versions and the translation of the mapping constraints into adaptation transformations could use the algorithm presented in this paper.

References


URL: http://exolutio.com


URL: http://www.omg.org/docs/omg/03-06-01.pdf

URL: http://www.xml.gr.jp/relax/


URL: http://www.omg.org/spec/UML/2.1.2/

Object Constraint Language Specification (2012), OMG.
URL: http://www.omg.org/spec/OCL/2.3.1/

Using Schematron as Schema Language in Conceptual Modeling for XML

Soběslav Benda Jakub Klímek Martin Nečaský
XML and Web Engineering Research Group Charles University in Prague Malostranské náměstí 25, 118 00 Praha 1, The Czech Republic Email: klimek@ksi.mff.cuni.cz

Abstract

Today, XML is a standard for message exchange inside and among IT infrastructures. For the exchange to work an XML format must be negotiated between the communicating parties. The format is often expressed as an XML schema. In our previous work, we introduced a conceptual model for XML, which utilizes modeling, evolution and maintenance of a set of XML schemas and allows exporting modeled formats into grammar-based XML schema languages like DTD and XML Schema. However, there is another type of XML schema languages called rule-based languages with Schematron as their representative. Expressing XML schemas in Schematron has advantages over grammar-based languages and in this paper, we identify the advantages and we propose a method for easier creation and maintenance of Schematron schemas using our conceptual model. Also, we discuss the possibilities and limitations of translation from our grammar-based conceptual model to the rule-based Schematron.

Keywords: XML schema, conceptual modeling, Schematron, translation

1 Introduction

Today, XML has many applications in various IT infrastructures. When using XML, communication partners must agree on the used XML formats, i.e. which elements and attributes may be present, in which order, etc. A specification of an XML format is an XML schema - a collection of rules which XML documents must satisfy. Programs that can automatically verify document validity are called validators. There is a number of declarative languages called XML schema languages used for description of schemas. The aim of these languages is to simplify the creation, maintenance, readability and portability of schemas.

The standardized schema languages are Document Type Definition (DTD), W3C XML Schema and REgular LAnguage for XML Next Generation (Relax NG). These languages have differences in some features (e.g. expressive power, syntax complexity, object-oriented design, etc). A common feature of these languages is their formal background which is a Regular Tree Grammar (RTG) (Murata et al. 2005). RTG determines the maximum expressive power and gives instructions for the construction of validators. Commonly, we call these languages grammar-based schema languages or grammars for short.

However, it is possible to express XML schemas in other languages that are not based on RTG. An example of such language is an also standardized Schematron (Jelliffe 2001). Briefly, Schematron allows describing schemas using XPath conditions, that are evaluated over a given XML document during validation. This brings on-demand testing possibilities for the validation of XML documents.

Motivation In our previous work (Nečaský, Mýlnková, Klímek & Malý 2012, Nečaský, Klímek, Malý & Mýlnková 2012), we developed a methodology for modeling, evolution and maintenance of XML schemas using a multilevel conceptual model based on Model Driven Architecture (MDA) (Miller & Mukerji 2003) and we introduced many extensions (Nečaský, Mýlnková & Klímek 2011, Klímek & Nečaský 2011) and described several use cases (Nečaský, Klímek & Malý 2011) of our approach. So far, we have only supported grammar-based XML schema languages, because of their popularity due to understandable declarations and efficient validation. While it is true that for relatively simple schemas DTD will do and for more complex structures XML Schema will provide the necessary constructs, there are also drawbacks to these widely used languages. For example, when we validate documents using DTD or XML Schema, we usually get a simple invalid statement as a result. In the more interesting case of invalidity, the validators usually return a built-in error message, which is often incomprehensible, misleading and does not provide means for quality diagnostics (Nálevek 2010). In addition, it is often not possible (or user-friendly) to pass them directly to the user interface. Regarding this diagnostic problem, Schematron schema can help. Schematron is often described as a language for description of integrity constraints (Murata et al. 2005), but it is more than that. Using Schematron, it is possible to describe most constraints that can be expressed by grammars. Moreover, it is possible to describe many additional details and even structural constraints that we can not express using grammar-like languages like XML Schema. In (Jelliffe 2007), the authors identify the demand for Schematron-based solutions for XML schema management, which is another motivation for adding support for Schematron to our conceptual model. Finally, when combined with the approach to express integrity constraints in the conceptual model (Malý & Nečaský 2012), Schematron becomes a unified schema language for description of the structure and integrity constraints of XML documents and a framework for detailed diagnostics and error reporting. These advantages of using Schematron outweigh its main disadvantage, which is its...
verbosity and complexity, because it can be eased by the usage of our conceptual model for schema management.

Outline The paper is organized as follows: In Section 2, we introduce our conceptual model for XML. In Section 3, we introduce the Schematron language. Section 4 contains the main contribution of this paper, the translation from the conceptual model to Schematron schemas. In Section 5 we discuss related work, in Section 6 we evaluate our approach and we conclude in Section 7.

2 Conceptual modeling of XML schemas

In this section, we briefly introduce our conceptual model for XML (see (Nečaský, Mlýnková, Klímek & Malý 2012) for full description). It is based on two levels of abstraction. The Platform-Independent Model (PIM) models the problem domain independently of any target platform such as XML or relational databases. The Platform-Specific Model (PSM) then provides description of how a part of the problem domain is represented in the target platform, in our case XML. A PSM schema is therefore a description of an XML format. From a PSM schema, we can automatically create a representation of the format in a chosen XML schema language such as XML Schema (our previous work (Nečaský, Mlýnková, Klímek & Malý 2012)), or, in the case of this paper, Schematron. The main feature of the conceptual model is a mapping, which specifies for each component in each PSM schema to which component in the PIM schema it corresponds. We exploit this mapping for automatic propagation of changes between the two levels, which simplifies the management of multiple XML schemas (Nečaský, Klímek, Malý & Mlýnková 2012).

2.1 Platform-Independent Model

A PIM schema $S$ is based on UML class diagrams and models real-world concepts and relationships between them independently of the target platform (implementation). It contains three types of components: classes, attributes and associations with the usual semantics. A sample PIM schema is in Figure 1(a). Formally, we define its simplification in Definition 1.

Definition 1 A platform-independent (PIM) schema is a triple $S = (S_c, S_a, S_r)$ of disjoint sets of classes, attributes, and associations, respectively.

- Class $C \in S_c$ has a name assigned by function name.
- Attribute $A \in S_a$ has a name, data type and cardinality assigned by functions name, type, and card, respectively. Moreover, $A$ is associated with a class from $S_c$ by function class.
- Association $R \in S_r$ is a set $R = \{E_1, E_2\}$, where $E_1$ and $E_2$ are called association ends of $R$. $R$ has a name assigned by function name. Both $E_1$ and $E_2$ have a cardinality assigned by function card and are associated with a class from $S_c$ by function participant. We will call participant($E_1$) and participant($E_2$) participants of $R$. name($R$) may be undefined, denoted by name($R$) = $\lambda$.

For a class $C \in S_c$, we will use attributes (C’) to denote the set of all attributes of $C$. Similarly, associations (C’) will denote the set of all associations with $C$ as a participant.

2.2 Platform-Specific Model

The platform-specific model (PSM) specifies how a part of the reality modeled on the PIM level is represented in the target platform, XML in our case, which makes the PSM schemas views of the PIM schema. Its advantage is that the designer works in a UML-style way which is more comfortable than editing the XML schema itself and also enables the maintenance of mappings to the PIM level. The individual constructs on the PSM level are, however, slightly modified to reflect the structure of XML documents. A PSM schema represents an XML format and can be automatically translated to the XML Schema language. Very briefly, classes represent complex types, attributes represent XML attributes or XML elements that have simple types, associations represent the nesting relation. For full translation description see our previous work (Nečaský, Mlýnková, Klímek & Malý 2012). Formally, PSM schema is defined by Definition 2. An example is in Figure 1(b). In Figure 1(c) there is a sample XML document with its structure modeled by the PSM schema.

Definition 2 A platform-specific (PSM) schema is a tuple $S’ = (S’_c, S’_a, S’_r, S’_m, C’_{S’})$ of disjoint sets of classes, attributes, associations, and content models, respectively, and one specific class $C’_{S’} \in S’_c$ called schema class.

- Class $C’ \in S’_c$ has a name assigned by function name.
- Attribute $A’ \in S’_a$ has a name, data type, cardinality and XML form assigned by functions name, type, card and xform, respectively. xform(A’) \in \{e, a\} (element or attribute). Moreover, it is associated with a class from $S’_c$ by function class and has a position assigned by function position within the attributes associated with class (A’).
- Association $R’ \in S’_r$ is a pair $R’ = (E’_1, E’_2)$, where $E’_1$ and $E’_2$ are called association ends of $R’$. Both $E’_1$ and $E’_2$ have a cardinality assigned by function card and each is associated with a class or content model from $S’_c$ or $S’_m$ assigned by function participant, respectively. We will call participant($E’_1$) and participant($E’_2$) parent and child and denote them by parent($R’$) and child($R’$), respectively. Moreover, $R’$ has a name assigned by function name and has a position assigned by function position within the associations with the same parent($R’$), name($R’$) may be undefined, denoted by name($R’$) = $\lambda$.
- Content model $M’ \in S’_m$ has a content model type assigned by function cmtype. cmtype($M’$) \in \{sequence, choice, set\}.

The graph $(S’_c \cup S’_m, S’_r)$ must be a forest of rooted trees with one of its trees rooted in $C’_{S’}$. For $C’ \in S’_c$, attributes (C’) will denote the sequence of all attributes of $C’$ ordered by position, i.e. attributes (C’) = $\{A’_i \in S’_a : \text{class}(A’_i) = C’ \land i = \text{position}(A’_i)\}$. Similarly, content (C’) will denote the sequence of all associations with $C’$ as a parent ordered by position, i.e. content (C’) = $\{R’_i \in S’_r : \text{parent}(R’_i) = C’ \land i = \text{position}(R’_i)\}$. We will call content (C’) content of $C’$.

2.3 Interpretation of PSM schema against PIM schema

A PSM schema represents a part of a PIM schema. A class, attribute or association in the PSM schema may be mapped to a class, attribute or association in the PIM schema. The mapping specifies the semantics of classes, attributes and associations of the PSM schema in terms of
the PIM schema. The mapping must meet certain conditions to ensure consistency between PIM schemas and the specified semantics of the PSM schema. This mapping is then utilized in various interesting use cases for the conceptual model like XML schema evolution and integration (Klimek & Nečaský 2010, Klímek & Nečaský 2011, Nečaský, Klímek, Malý & Mlýnková 2012). For the precise conditions of the mapping, see (Nečaský, Mlýnková, Klímek & Malý 2012). In this paper, we focus on the translation from a PSM schema to Schematron and therefore, the precise definition of interpretation is not necessary here.

2.4 Conceptual model summary

In summary, the usefulness of our conceptual model for XML can be clearly seen when we, for example, ask questions like “In which of our hundred XML schemas used in our system is the concept of a customer represented?” and “What impact on my XML schemas would this particular change on the conceptual level have?”. Even better, with our extensions for evolution of XML schemas (Klimek & Nečaský 2010, Nečaský, Klímek, Malý & Mlýnková 2012) we can make changes to the PIM schema (e.g. change the representation of a customer’s name from one string to firstname and lastname) and those changes can be automatically propagated to all the affected PSM schemas. Thanks to automated translations from PSM schemas to, for example, XML Schema and back (Nečaský, Mlýnková, Klímek, Malý & Mlýnková 2012), we can easily manage a whole system of XML schemas from the conceptual level all thanks to the interpretations. These extensions, however, are not trivial and are not in the scope of this paper. Also, it would be possible to generate a clickable HTML documentation of a system modeled using our conceptual model. With the model, it is also much easier and faster to grasp a system of multiple XML schemas when, for example, negotiating interfaces between two information systems. We already have our model implemented in our tool called eXoluto.3

3 Schematron

Schematron is a declarative language which represents the rule-based XML schema languages. These languages are not based on construction of a grammatical infrastructure. Instead, they use rules resembling if-then-else statements to describe constraints. These languages offer the finest granularity of control over the format of the documents (Vlist June 2002). We can even view constructs of other schema languages as syntactical sugar instead of sets of rule-based conditions. Schematron was designed in 1999 by Rick Jelliffe. The language was standardized in 2005 as ISO Schematron (Jelliffe 2001).

Example 1 Consider a specification of a complex element using the following DTD declaration

```xml
<!ELEMENT purchase (item+,customer?)>
```

In Schematron, it is possible to describe the same semantics and cover valid instances using multiple intuitive conditions, for example: If purchase element exists, the element can only have an item and a customer elements as children. The item element has at least one occurrence and the customer element has zero or one occurrence. If the customer element exists as a child element of a purchase element, then the customer element has no following sibling elements.

Schematron is not a standalone language. It is a general framework which allows schema designers to organize conditions which are evaluated over the given documents. These conditions are described using an underlying XML query language such as the default XPath. A result of a validation is a report which contains information about evaluation of these conditions. Schematron is an XML-based language and uses only few elements and attributes for schema description.

3.1 Core constructs

Now we describe core Schematron constructs. The root element of every schema is a schema element introducing the required XML namespace. A pattern element is a basic building stone for expressing an ordered collection of Schematron conditions which are ordered in XML document order. A rule is a Schematron condition which allows a designer to specify a selection of nodes from a given document and evaluation of predicates in the context of these nodes. The rule element has a required context attribute used for an expression in the underlying query language. The value of the context attribute is commonly called a path. Predicates are specified using a collection of assertions. An assertion is a predicate which can be positive or negative. An assertion is represented using the assert and report elements. Both elements have a required test attribute for specification of a predicate using the underlying query language. Both elements also have a text content called natural-assertion. Natural-assertion is a message in a natural language, which a validator can return in the validation report. A positive predicate is represented using an assert element and if it is evaluated as false, we say that the assert is violated and the document is invalid. A negative predicate is represented using a report element and if it is evaluated as true, we say that the report is active.

---

3http://exoluto.com

4http://purl.oclc.org/dsdl/schematron
and a natural-assertion will be reported. Schematron is not only a validation language. It is a more general XML reporting language (Ogbuji 2004) where one type of report is an error message.

**Example 2** A pattern in Figure 2 selects all triangle elements from a document. In a context of every triangle element a positive predicate specified with expression count (vertex) = 3 is evaluated. If the given

```xml
<pattern>
  <rule context="triangle">  
    <assert test="count (vertex) = 3">  
      The element ’triangle’ should have 3 ’vertex’ elements.  
    </assert>  
  </rule>
</pattern>
```

Figure 2: Schematron pattern

triangle has for example four child vertex elements, then the predicate will be false and the following message will be reported: The element ’triangle’ should have 3 ’vertex’ elements.

### 3.2 Additional constructs

In addition to the core Schematron constructs we describe other constructs used in practical applications. Schematron allows using metadata for introduced constructs. Identifiers allow identification of a pattern inside a schema, a rule inside the pattern, etc. It is represented using an id attribute. A role attribute can be used to assign special semantics to Schematron constructs. A diagnostic is a natural-language message giving details about a failed assertion, such as found versus expected values and repair hints. It is represented using a diagnostic element with required id attribute and text content with a message. Diagnostics are referenced by assertions using a diagnostics attribute. We can use substitutions in natural-assertions for clearer result in validation reports. An element name is substituted by the name of a context node. An element value-of is substituted by a value found or calculated using an expression in the required select attribute. Abstract rules provide a mechanism for reducing schema size. An abstract rule can be invoked by other rules belonging to the same pattern. Variables are substituted in assertion tests and other expressions before the expression is evaluated. Phases allow to organize patterns into identified parts. Every Schematron schema has one default phase which includes all patterns. Before validation, it can be determined which phase is used and which patterns are activated. This selected phase is called an active-phase. A phase is represented using a phase element with an id attribute. One phase can have multiple active elements which refer to patterns using a pattern attribute. A variable is represented using a let element. If the variable is a child of a rule, the variable is calculated in scope of the current rule and context. Otherwise, the variable is calculated within the context of the instance document root. Abstract patterns allow a common definition mechanism for structures which use different names and paths, but which are the same otherwise.

### 3.3 Schematron implementations

An implementation of Schematron validation is very simple in general, because it is based on already implemented XML technologies. There are two kinds of Schematron validation.

#### 3.3.1 XSLT validation

For this approach, we only need an XSLT processor and a predefined XSLT script\(^4\). The script translates the given Schematron schema to another XSLT script which is used for the actual XML document validation. During the validation, a given XML document is transformed into another XML document. This document is the result of the validation and may be formatted using standard Schematron Validation Report Language (SVRL) (Jelliffe 2001), which provides rich information about the validation process, e.g. XPaths for elements which violated assertions.

#### 3.3.2 Special libraries

Another approach is to use a special (platform-dependent) library. Some libraries\(^4\) only wrap the described XSLT validation. However, there are other implementations not based on XSLT. These libraries are based on the evaluation of XPath expressions. This allows a programmer to adapt the validation for special requirements or possibilities of a target platform. For example, we have implemented a C# validator called SchemaTron (Benda et al. 2011) providing excellent performance for XML content-based message routing inside an intermediate service.

### 3.4 Schematron properties

In this section, we describe selected properties of Schematron schemas in the context of conceptual modeling of XML and compare them with grammar-based schemas. We mostly consider XML Schema 1.0 as their representative because we already support it in our conceptual model for XML.

#### 3.4.1 Platform independence

Schematron is based on standard XML technologies, which are commonly implemented in many software environments, e.g. XSLT processor is natively implemented in standard web browsers. For these reasons, we can see Schematron as a platform independent XML schema language, because we do not need a specific validator.

#### 3.4.2 Expressive power

Presently, there is no formal framework (Kwong & Gertz 2006) which could capture the broad set of possibilities of Schematron conditions.

The authors of (Lee & Chu 2000) provide some basic expressive possibilities of Schematron, compare it with other schema languages and show by example that Schematron has an excellent expressive power and in this regard it is (e.g. with XSD) a first class XML schema language. The authors describe, that we can specify for example: parent-child relationships, sequences, choices among elements and attributes, unordered sets, min and max occurrences, etc. Moreover, we can specify many XML formats, which can not be expressed using grammars, for example conditional definitions (e.g. a presence of elements or attributes is dependent on a value of another element or attribute) or detailed integrity constraints, etc. However, there is not any precise generalization of Schematron rules which would provide a clever mapping of regular tree grammar into Schematron rules (and vice versa), but experiments show (Jelliffe 2007) that it is possible to describe many instances of grammars in Schematron, even in diverse ways.

---

\(^{3}\)http://www.schematron.com/tmp/iso-schematron-xslt1.zip

\(^{4}\)http://www.probatron.org/
4 PSM to Schematron translation

A PSM schema models a grammar-based XML format specification and its concepts are interpreted against PIM concepts. There are several theoretical and practical problems that we must consider when we want to describe the translation of a PSM schema to a Schematron schema. In particular, we need to identify groups of Schematron rules that impose equivalent constraints on the documents as constructs of grammar-based languages would.

4.1 Overall view of the translation

The translation algorithm (see Algorithm 1) is fully automatic. It has a PSM schema on the input and it gradually builds a Schematron schema on the output. The generated schema covers grammatical structural constraints normally expressed in, for example, XSD.

Algorithm 1 Overall view of the translation algorithm

1. Generate allowed root element names (Section 4.2);
2. Generate allowed names (Section 4.3);
3. Generate allowed contexts (Section 4.4);
4. Generate required structural constraints (Section 4.5);
5. Generate required text restrictions (Section 4.7);
6. Generate required sibling relationships (Section 4.6);
7. Generate required text restrictions (Section 4.7);
8. Generate required text restrictions (Section 4.7);
9. < /schema >

In the first step (line 2), we generate Schematron patterns for XML elements, which are allowed inside valid XML documents as root elements. Similarly, we generate patterns for allowed names of XML elements and XML attributes in the next step on lines 3. On line 4, we produce patterns for allowed contexts, i.e., paths where certain names of elements (and attributes) may occur. We call the generated patterns absorbing patterns and we describe them later. The patterns for validation of required complex element structures are produced in the steps on lines 5 and 6. These patterns are more complex, because we must generate an equivalent of regular expressions to obtain the semantics of regular grammars. We call these patterns conditional patterns and we describe them later in this section. In the last step (line 7), the patterns for text restrictions, i.e., validation of attribute values and simple element contents, are produced.

4.2 Allowed root element names

We need a tool for reporting names of elements which are not allowed in the schema, but are present in the document.

Definition 3 An absorbing pattern is a Schematron pattern for an ordered set of paths P, where the last rule is called global and it contains the * wildcard somewhere in its path and no previous rules use wildcards.

In this definition we defined a special kind of a Schematron pattern which we call an absorbing pattern and which allows Schematron to absorb elements (or attributes) specified by paths. It checks for all the allowed elements or attributes in the path and if none of them is found, it matches whatever is found in the path using a wildcard (absorbs it), so that the validation can continue. If the element or attribute is absorbed by the wildcard, it is a violation of the expected format and the element or attribute absorbed by the wildcard rule is reported. In the first step on line 2 in the overall Algorithm 1, we generate an absorbing pattern for checking allowed root elements as the intuitive way described here.

Example 3 As an example, consider the set of paths P, which contains paths for all allowed root elements /request and /response. We generate the absorbing pattern that is in Figure 3. When the validated document

\[
\langle \text{pattern id="allowed-root-elements" role="absorbing-pattern"} \rangle
\]

\[
\langle \text{rule context="/request"} \rangle
\]

\[
\langle \text{assert test="true()"} \rangle
\]

\[
\langle \text{rule context="/response"} \rangle
\]

\[
\langle \text{assert test="true()"} \rangle
\]

\[
\langle \text{rule context="/"} \rangle
\]

\[
\langle \text{assert test="false()"} \rangle
\]

The element '<name/>' is not declared in the schema as a root element. However, the validation still continues, which is in contrast to XSD validation, which would end at this point.

Figure 3: Absorbing pattern example

4.3 Allowed names

The approach is very similar to generation of allowed root element names, because we also generate absorbing patterns. Patterns for allowed attributes are also similar, so we skip them in this paper. Production of patterns for checking allowed XML elements inside validated documents follows this algorithm: We produce the set P of all paths for allowed element names. For complex elements, we get them from the names of named associations which have classes as children in the PSM schema. For simple elements, we get the names from PSM attributes A' with XML form set to xform(A') = ε. From the paths and names, an absorbing pattern is generated.

4.4 Allowed contexts

Now we introduce stricter patterns for checking allowed contexts, i.e., paths inside documents. We also generate absorbing patterns, but we need more sophisticated paths, because we absorb only element and attribute names in the declared contexts, so the other names (contexts) break validity.

Paths overview A path is described using an XPath expression to select some nodes from the validated XML document. When nodes are selected, we can evaluate assertions, i.e., certain XPath predicates in the context of these nodes. In general, we have two approaches to how we can describe paths, i.e., absolute paths, for example /book/author/name or relative paths for example name. If we want to design schemas more powerful than DTD, i.e., local regular tree grammars (Murata et al. 2005), we need absolute paths to select nodes from documents. However, relative paths are also important for example to
design recursive declarations. There is also a possibility to use predicates in paths. We do not deal with predicates for now, although because we aim to design a Schematron schema as simple as possible. For this purpose, we impose a SORE precondition on our PSM schemas in Definition 4. Every SORE is deterministic as required by the XML specification and more than 99% of the regular expressions in practical schemas are SORES (Bex et al. 2006), so the precondition does not limit us much and at the same time simplifies the translation a lot. For instance, \((a|b).c\) 0..+,\(d\) 0..1) 0..3 is SORE while \(a(a|b)\) 0..* is not as occurs twice.

**Definition 4** Let \(S'\) be a PSM schema. We will call SORE precondition an assumption on \(S'\), that every complex element has content described using Single Occurrence Regular Expression, i.e. every element (or attribute) name can occur at most once in this regular expression.

**Paths construction** Here we describe the construction of paths for a PSM schema. The main idea is as follows. For each XML element and XML attribute declaration present in a PSM schema, we produce all possible paths (contexts) where they can occur. Every created path is associated with a PSM component, i.e. a complex element, a simple element or an attribute declaration and the pairs are placed into the global set of paths \(G_p\). In the next step, we perform sorting of \(G_p\) members. The resulting ordered set \(G_p = \{(X', p) ; X' \in (S'_u \cup S'_a)\} p\) is used for generation of Schematron rules in the order of this set in the rest of the translation. We sort members of \(G_p\) using the following ordering: (1) The absolute paths without recursions go first (2) The absolute paths with recursions follow, the longest path is the first one (3) The relative paths go last, again, the longest path is the first one to go.

Firstly, we need to create all paths for a given XML element or XML attribute declaration. Let us mark the declaration – the given PSM component \(X' \in (S'_u \cup S'_a)\). We build an ancestor tree for \(X'\) which represents all achievable ancestor PSM components of \(X'\) in the PSM schema. Then we can translate all its paths from leaf nodes to root node into Schematron paths, i.e. XPath expressions. For each \((X', p) \in G_p\), we must hold that \(p\) is unique, which corresponds to the SORE precondition in Definition 4.

**Pattern for allowed element contexts** Now we can produce patterns for allowed contexts. We go through all members of the ordered set \(G_p\) and produce set of paths \(P\) only for complex element names and simple element names (PSM attributes with XML form set to element). In the last step we produce an absorbing pattern for \(P\) with \(\ast\). Similarly, we produce a pattern for allowed attribute contexts.

**4.5 Required structural constraints**

Now we have absorbing patterns for weak validation of XML documents generated. These patterns say what is allowed inside the documents. Now we deal with restrictions which say what the given document must satisfy.

**Conditional pattern** First of all, we specify another Schematron pattern, which we call conditional pattern (see Definition 5).

**Definition 5** A conditional pattern is a Schematron pattern for a set of pairs \(E = \{(p, A) ; p\) is a path and \(A\) is a set of predicates\}. It consists of several rules and the document passes validation by this pattern only if all the rules are satisfied.

**Example 4** Consider a PSM schema where the root element customer must have a ship-to child element which must have a street child element and the street must not have any child elements. We generate a conditional pattern, which is in Figure 4. The pattern re-

```
<schema>
  <element name="customer">
  <complexType>
    <sequence>
      <element name="ship-to">
        <complexType>
          <sequence>
            <element name="street"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
</schema>
```

![Figure 4: Conditional pattern example](image)

sembles a collection of if-then conditions, because it says: If the customer element exists in the document as a root, it must hold that it has a ship-to child element. If the ship-to element exists in the document as a child of the customer element, it must hold that it has a street child element, etc. The example demonstrates, that we can create such a pattern with chained rules. If we wanted to describe this pattern using XML Schema, it would be:

```
<element name="customer">
  <complexType>
    <sequence>
      <element name="ship-to">
        <complexType>
          <sequence>
            <element name="street"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
```

For the production of conditional patterns, we need to analyze specifications of complex element contents. The complex element declared in a PSM schema is precisely specified using a regular expression, so we need to analyze such regular expressions and translate them into Schematron predicates. The main idea is as follows. We translate the regular expression into several conditional patterns. These patterns cover the same semantics as the regular expression, when they are evaluated together.

We generate two conditional patterns for checking structural constraints as a part of Algorithm 1 in the step on line 5. One of the generated patterns checks required parent-child relationships, the other pattern checks required parent-attribute relationships and other relationships of attributes and elements (predicates for choices among attributes and elements). There can be also other distributions of conditions into patterns. For example, everything may be inside one pattern, but we believe that our distribution provides flexible solution, because we can check required elements only, required attributes only, etc. In the algorithm, we use translations of regular expressions into boolean expressions and then normalization of boolean expressions into conjunctive normal form (CNF). Due to lack of space and because these are quite standard transformations, we do not describe them here in detail.
Algorithm 2 Generate patterns for structural constrains
1: Let \( E_1 \) be an empty set of pairs \((p, A_p)\);  
2: Let \( E_2 \) be an empty set of pairs \((p, A_p)\);  
3: for all \((X', p) \in G_p\) do  
4:   if \( X' \in S' \) then  
5:       Let \( A_p \) be an empty set of predicates;  
6:       Let \( A_p \) be an empty set of predicates;  
7:       for all \( Y \in \text{cnf}(be'(X'))\) do  
8:          if \( Y \) has only elements in its literals then  
9:             Add \( Y \) into \( A_e \);  
10:            else  
11:                Add \( Y \) into \( A_e \);  
12:            end if  
13:       end for  
14:       if \( A_e \) is not empty then  
15:           Add \( (p, A_e) \) into \( E_1 \);  
16:       end if  
17:       if \( A_e \) is not empty then  
18:           Add \( (p, A_e) \) into \( E_2 \);  
19:       end if  
20:   end if  
21:  end for  
22: Generate conditional pattern for \( E_1 \);  
23: Generate conditional pattern for \( E_2 \);  

Let us now take a look at Algorithm 2 for production of patterns for structural constrains based on boolean expressions. Firstly, we initialize two empty sets of pairs \((p, A_p)\) (line 1) and \((p, A_p)\) (line 2), where \( p \) is a path and \( A_p \) is a set of associated predicates for elements, \( A_p \) is a set of associated predicates for attributes and relations with elements. Then, we go through pairs \((X', p) \in G_p\) and when \( X' \) is a complex element declaration, we initialize new sets \( A_e \) and \( A_e \) (lines 5 and 6) and translate \( X' \) into boolean expression and the boolean expression into conjunctive normal form (line 7). Then, we go through obtained predicates. When a predicate (marked as \( Y \)) has only elements in its literals we add it into \( A_e \), else we add it into \( A_e \). Then, if \( A_e \) or \( A_e \) is not empty, we add a pair \((p, A_e)\) or \((p, A_e)\) into \( E_1 \) (line 15), or \( E_2 \) (line 18), respectively. In the last step (lines 22 and 23), we generate conditional patterns for \( E_1 \), \( E_2 \). We have created two patterns, which cover certain structural constrains of modeled complex contents.

Example 5 As an example, consider a regular expression which specifies the complex element item in Figure 5(a): \((@\text{code}, (\text{amount}, \text{price}))/@\text{tester})\). We translate it into \@code and (\text{amount and price and count}(@\text{tester})=0) or (\text{count}(@\text{amount, price})=0 and @\text{tester})\), which is an XPath predicate that we use in Schematron assertion in Figure 5(b). This representation is quite straightforward and corresponds well with grammar-based languages like XML Schema. However, it also comes with disadvantages in the form of poor diagnostics. As with XML Schema validation, when we would validate a document using the Schematron rule from Figure 5(b), we would only get a valid or invalid statement without further details. For this purpose, it is more advantageous to go into more detail and write the same rule as multiple simpler rules. We transform the regular expression, which is a logic formula, to a conjunctive normal form as seen in Figure 5(c). Now, we can create user-friendly diagnostics for each of these rules. Note that this is also an example of choice between attributes, which is not possible in XML Schema but can be done using Schematron.

4.6 Required sibling relationships
In the previous section we generated structural constraints using boolean expressions, which allow to validate parent-child relationships. So far we did not deal with the order of child elements inside a parent element. Here we describe our approach based on the theory of regular expressions. The main idea is as follows. We build a finite state automaton for a given regular expression. We deal only with SRE so we can build the deterministic SRE automaton, where every name of XML element is assigned to at most one inner state and it has one initial and one final state. Then we translate information obtained from this structure into Schematron conditions. We represent the transition function of the automaton using conditional patterns and we cover for example the order of XML elements (sequences, choices among elements) and also cardinalities zero or one \((0..1)\), or \(?,\) just one \((1..1)\), zero or more \((0..*\), or Kleene star \(*\), one or more \((1..*\), or Kleene cross \(+\). We can also provide clear natural-assertions and diagnostics.

There are also some problems and exceptions. Firstly, we can not cover arbitrary numeric intervals of regular expressions using this approach (it is possible to create an automaton with numeric intervals, but it is not possible to represent it in Schematron). We need another approach for numeric constrains in general, which is not part of this paper. Secondly, a PSM content model SET complicates construction of the algorithm. The restriction (Definition 6) for content model SET is similar to restriction of XSD construct ALL.

Definition 6 Let \( S' \) be a PSM schema. We introduce SET precondition, which is an assumption on \( S' \), that for
each content model \( S' \) it must hold that it has named associations with classes as children in its content and the content model is descendant of associations \( R' \in S'_s, \langle \text{name}(R') = \lambda \lor \text{child}(R') \notin S'_s \rangle \) in the complex content, where \( \text{card}'(R') = 0..1 \) or \( \text{card}'(R') = 1..1 \).

Now we can presume that we can build the automaton for each complex element declaration, i.e. named association with class as a child. We also need to translate obtained information into Schematron rules. For each complex element and for elements in its content we produce a set of predicates. These predicates are composed from following-sibling XPath axes. For each of the obtained predicates we generate a conditional pattern in the step on line 6 in Algorithm 1.

Example 6 Consider content \( \langle \text{title?}, \text{name}, \langle \text{phone} | \text{e-mail} \rangle + \rangle \). We can represent this regular expression using the SORE automaton in Figure 6(a). Then we generate Schematron rules (see Figure 6(b)) which represent the automaton in Schematron. Note that we use \( F := \text{following}-\text{sibling} \) substitution for code size reduction in this example. The first rule represents the initial state of the automaton and says what elements can be at the first position in the content. Other rules represent if-then conditions, i.e. if \( \text{title} \) element exists, it has a name follower. If name element exists, it has a phone or an e-mail followers. If phone element exists, it has the phone element or the e-mail element followers or no following-sibling elements.

4.7 Required text restrictions

In this section we show the final patterns of our proposed translation from a PSM schema of our conceptual model for XML to Schematron. They deal with validation of data types for simple element contents and attribute values. The supported set of data types for PSM attributes is implementation dependent, as we need the datatypes to be defined by the designer in Schematron. Our implementation, eXolutio, provides XSD built-in simple data types and we created the rules for their definition in Schematron, because Schematron does not provide built-in data types as default. We can, however, create many data types specifications using XPath expressions placed into abstract rules or patterns. Schematron over XPath 1.0, which we describe here, is worse than XSD in this practical aspect and we need to help ourselves by depending on the designer to define the used datatypes. Examples of these definitions are in Figures 7 - 11.

Definition 7 Let \( S' \) be a PSM schema. We will call data types precondition an assumption on \( S' \), that each data type used in \( S' \) has corresponding declaration in Schematron provided by the designer or by the eXolutio tool.

In the step on line 7 of Algorithm 1 we generate patterns for data types validation as extension rules of our predefined data type rules using the <extend/> element in the <rule/> element.

4.8 Translation summary

In this section, we introduced the problem of automatic construction of Schematron schemas from PSM schemas. The translation is not simple, because we have different models - grammar-based PSM schema (and XML Schema, DTD, etc.) and the rule-based Schematron. However, we showed that Schematron is a very powerful language and it can express many grammatical structural constrains from the grammar-based languages and more.

We started with production of absorbing patterns, which allow to validate allowed occurrences of XML elements and XML attributes inside validated XML documents. Then we produced conditional patterns for validation of required grammatical structural constraints. We analyzed the most used parts of regular expressions which can be represented in Schematron. Then we generated patterns for validation of data types for simple element contents and attribute values.

There are some limitations to our approach that, however, do not seem critical at the moment. The most visible one is the lack of support for arbitrary numeric intervals in cardinalities. We only support the usual 0..*, 0..1, 1..*, 1..1. This is because the support for arbitrary intervals would necessarily lead to Schematron code explosions which would only complicate and slow down the validation process.

5 Related work

In parallel to the research of translation of PSM schemas to Schematron, other PSM schema improvements are also being researched. In particular the support for Object Constraint Language (OCL) (Object Constraint Language Specification, version 2.0 2009) and its translation to Schematron for the specification of integrity constraints, where Schematron is used as a complement of grammar-based schemas. These patterns for integrity constraints generated from OCL may be potentially merged with our Schematron schemas. To our best knowledge, little work has been done in the area of translations between Schematron and other XML schema languages. There are sources not based on academic research which provide some basic ideas and techniques for translation of grammar-based schemas to Schematron schemas and vice versa. Most work in this area has been done by Rick Jelliffe and his
company Topologi\(^5\). They have implemented an XSD to Schematron converter\(^6\), because their customers preferred Schematron diagnostics over XSD validation. The generated schemas are called Schematron-ish grammars. In (Nečaský, Mlýnková, Klímek & Malý 2012), we provide formal description of mutual translation between PSM schemas and regular tree grammars.

6 Evaluation and implementation

With our proposed method, we have generated several Schematron schemas in various data domains using our conceptual model. The schemas are verbose and cannot be shown here whole due to space limitations. Their structure is, however, shown in our examples throughout the paper. During our experiments, we found the Schematron based validation as easy to use from a domain expert’s perspective as a validation using XML Schema would be given that both can be generated from our conceptual model for XML. The downside of Schematron mentioned in our motivation, which is its verbosity, is not a problem in the end because the user does not need to read the actual generated Schematron. He only needs to give it as an input to a Schematron validator. From the validation performance point of view, rule-based validation (e.g. Schematron) is computationally more expensive than the linear validation using grammar-based languages (such as XML Schema) (Nálevka 2010). This could be a problem in an environment that requires high performance validations, such as routing of XML messages. Nevertheless, when performance is not an issue or when validating against complex XML formats, the benefits in form of better diagnostics are more important.

The reward for using our approach is much easier diagnostics of a possible problem in the validated XML document because as we described in this paper, Schematron supports user-friendly and descriptive error messages. Also, its expressive power is greater than that of XML Schema, which can be seen in Figure 5, where we use a choice between attributes, which is not possible to express in XML Schema. Our experiments were done using our implementation of the conceptual model for XML, eXolutio.

eXolutio is an application developed in our research group. Its base is the formalism for our conceptual model for XML described in (Nečaský, Mlýnková, Klímek & Malý 2012) and a complex system of operations and their propagation between the levels of abstraction described in (Nečaský, Klímek, Malý & Mlýnková 2012). In addition, it is a platform where novel extensions to XML schema modeling and evolution are implemented. One of them is the approach described in this paper. In Figure 12 there is a PSM schema modeled in eXolutio and in Figure 13 there is the generated Schematron schema.

---

\(^5\) http://www.topologi.com/

\(^6\) http://www.schematron.com/resource/

XSD2SCH-2010-03-11.zip
7 Conclusions

In this paper, we briefly introduced our conceptual model for XML as a basis for modeling and maintenance of XML schemas independent of the target schema language. Then we introduced Schematron, a rule-based language that can be used for XML schema description, and its constructs. Next, we described how a schema from our conceptual model can be translated to Schematron and described the advantages over grammar-based languages such as XML Schema. We have evaluated our approach and described its implementation in our tool, eXolutio.

References


Bex, G. J., Neven, F., Schwentick, T. & Tuyls, K. (2006), Inference of Concise DTDs from XML Data, ACM.

URL: http://www.ascc.net/resource/schematron/


URL: http://www.balisage.net/Proceedings/vol8/html/Maly01/BalisageVol8-Maly01.html

URL: http://www.omg.org/docs/omg/03-06-01.pdf


URL: http://petrnalevka.blogspot.com/2010/05/grammar-vs-rules.html


Object Constraint Language Specification, version 2.0 (2009), OMG.
URL: http://www.omg.org/technology/documents/formal/ocl.htm

Ogbuji, U. (2004), A hands-on introduction to Schematron, IBM.

Vlist, E. (June 2002), XML Schema The W3C’s Object-Oriented Descriptions for XML, O’Reilly Media.
Abstract

Classical database theory is largely a theory of relations. Relations are sets of tuples in which no duplicate tuples occur. In practice, duplicate elimination is an operation that is considered to be too expensive in many situations. Fundamental classes of integrity constraints interact differently over bags than they do over relations. This holds for keys and functional dependencies, for example. In this paper, we establish algorithms that compute Armstrong bags for any given set of keys and functional dependencies. These are bags which satisfy the given set of keys and functional dependencies, but violate all keys and functional dependencies not implied by the given set. Armstrong bags perfectly visualize abstract sets of constraints, and can be used by database designers to effectively communicate perceptions or states of a database. Subsequently, we show how existing algorithms to discover functional dependencies in relations can be adapted to discover keys and functional dependencies in bags.

Keywords: Armstrong database, Bag, Constraint, Cover, Discovery, Functional dependency, Key

1 Introduction

A database system is a software package that manages a collection of persistent information in a shared, reliable, effective and efficient way. Most database systems are still founded on the relational model of data (Codd 1970). In this model, data is stored in a collection of relations that may vary over time. A relation is a set of tuples over a given time-invariant relation schema. The relation schema itself is a set of attributes that model all the properties that each tuple of each relation over the schema is described by. That is, a tuple maps each attribute of the relation schema to a value from the domain (a set of possible values) of this attribute. One may think of a relation as a table in which the column headers are given by the attributes of the relation schema, and each row of the table is given by a tuple.

There is a gap between database theory and practice: all commercial relational database management systems allow duplicate tuples, and so relations in such systems are in effect bags. That is, duplicate tuples usually occur in real world database instances, even though the relational model of data is set-oriented, i.e., does not allow duplicate tuples to occur. Indeed, databases and bags are tightly coupled. For instance, the cost of duplicate removal, e.g. after projections of relations or the union of several relations, is frequently considered to be too expensive. Therefore, duplicate removal is only performed if the user explicitly requests so. Duplicate detection has evolved into its own area of research in recent years (Naumann & Herschel 2010). Furthermore, bag processing has applications in the evaluation of database queries, and in areas such as view maintenance, data warehousing and web information discovery. Consequently, there has been an effort in database research to establish a foundation for handling bags instead of sets, cf. (Lamperti et al. 2000).

1.1 Background and Motivation

Surprisingly, an extension of the relational framework of data dependencies to bags has not received much attention. Data dependencies are specified as semantic constraints on a relation schema that restrict the set of possible database instances to those which are considered meaningful to the application domain at hand. The intuitive meaning of “dependency” is that the occurrence of data values satisfying a set of certain properties enforces some properties of other data values. In this sense, the latter values are “dependent” on the former ones. A prime example is given by the class of functional dependencies (FDs). These are expressions of the form \( X \rightarrow Y \) where \( X \) and \( Y \) are subsets of the same relation schema \( R \). An instance over \( R \) satisfies this functional dependency whenever any two tuples of the instance agree on all attributes of \( X \), then they also agree on all the attributes of \( Y \). In this sense, the values on attributes of \( Y \) are functionally dependent on the values on attributes of \( X \). FDs may imply other FDs, i.e., whenever a relation satisfies a set of FDs, then it may also satisfy other FDs. The implication problem of FDs is to decide for an arbitrary relation schema and an arbitrary set \( \Sigma \cup \{ \varphi \} \) of FDs on the relation schema, whether \( \Sigma \) implies \( \varphi \). The finite axiomatisability, i.e. the existence of a finite, sound and complete set of syntactical inference rules, and the time-complexity of the implication problem for FDs in relational databases has been well-studied in the literature (Armstrong 1974, Diederich & Milton 1988). In particular, the implication of FDs is equivalent to the implication of Horn clauses in Boolean propositional logic (Fagin 1982b).
and the pullback rule. Logically, FDs correspond to
ular, the superkey rule follows from reflexivity axiom
mits sets of tuples. In previous work (Köhler & Link
but the underlying relational model of data only per-
database management systems permit bags of tuples,
acterized when bags are permitted as database in-
dependency Item, Size
Example 1 Consider a database where we store cus-
ter orders, e.g., for Japanese cuisine. We record the Item, and the Size and Price of each Item. An order may look as follows:

<table>
<thead>
<tr>
<th>Item</th>
<th>Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>takoyaki</td>
<td>large</td>
<td>¥90</td>
</tr>
<tr>
<td>takoyaki</td>
<td>large</td>
<td>¥90</td>
</tr>
<tr>
<td>nigiri sake</td>
<td>large</td>
<td>¥90</td>
</tr>
<tr>
<td>nigiri sake</td>
<td>average</td>
<td>¥70</td>
</tr>
<tr>
<td>okonomiyaki</td>
<td>small</td>
<td>¥300</td>
</tr>
</tbody>
</table>

For this bag, no subset of \{Item, Size, Price\} is a key. In particular, it does not satisfy the key \(k\{Item, Size\}\). However, it does satisfy the functional dependency \(Item, Size \rightarrow Item, Size, Price\).

Consequently, it is a natural question to ask how the implication problem of keys and FDs can be char-
acterized when bags are permitted as database in-
stances. The question is significant since relational
database management systems permit bags of tuples,
but the underlying relational model of data only per-
mits sets of tuples. In previous work (Köhler & Link
2010) we have characterized the implication problem
axiomatically, algorithmically, and logically. For ex-
ample, the inference rules from Table 2 form a mini-
mal axiomatization (Köhler & Link 2010). In partic-
ular, the superkey rule follows from reflexivity axiom
and the pullback rule. Logically, FDs correspond to
definite Horn clauses, and keys to goal Horn clauses;
refining the correspondence between FDs and Horn
clauses over relations (Köhler & Link 2010).

### 1.2 Contributions

In this paper, we are interested in the discovery of
keys and functional dependencies. First, we will ad-
dress how to support the effective discovery of keys
and functional dependencies that are semantically
meaningful for a given application domain. In prac-
tice, there is a mismatch in expertise. Domain ex-
erts know a lot about a given application domain,
but do not know anything about database concepts.
Database designers know a lot about database con-
cepts, but do not know a lot about the application
domain for which they are hired to develop an infor-
mation system. This mismatch makes it difficult to
exchange perceptions about the application domain.
In particular, database designers and business ana-
lysists face the difficult task to acquire the constraints
perceived semantically meaningful by the domain ex-
perts. The task is difficult since database constraints
are quite abstract and their interaction is even harder
to communicate. Since humans can learn a lot from
eamples, data samples that faithfully represent ab-
tract sets of constraints could be a helpful tool for the
discovery of semantically meaningful constraints. The
notion of data samples that faithfully represent ab-
tract constraint sets is known as Armstrong samples
in the database literature (Fugin 1982a, Link 2012).
A data sample is Armstrong for a given set \(\Sigma\) of con-
straints in a given class \(C\), if the data sample satisfies
all constraints in \(\Sigma\) and violates all constraints in \(C\)
that are not implied by \(\Sigma\). Therefore, if \(\Sigma\) repres-
ts the set of constraints currently perceived meaningful
by a design team, then an Armstrong sample for \(\Sigma\)
satisfies all constraints currently perceived meaning-
ful, and violates all constraints currently perceived
meaningless. The simple and fundamental idea is that
a domain expert who inspects an Armstrong table
will easily spot violations of actually meaningful con-
straints that are currently perceived meaningless by
the design team. The domain experts can point these
violations out to the designers, who thus learn seman-
tically meaningful constraints. These intuitions about
the usefulness of Armstrong samples for the acquisi-
tion of semantically meaningful functional dependen-
cies have been empirically validated in the literature
(Langeveldt & Link 2010).

Example 2 Suppose the design team for the appli-
cation domain of Example 1 is certain that the FD
\(Item, Size \rightarrow Price\) is a semantically meaningful con-
straint. However, they are uncertain about other
FDs, and keys in general. To consolidate their un-
derstanding, they create the Armstrong sample for
\(\Sigma = \{Item, Size \rightarrow Price\}\) from Example 1 that they
show to some domain experts. When inspecting the
Armstrong bag they wonder why the price of ¥70 ap-
plies to both average and small sizes of nigiri sake.
After some discussion, it turns out that the same items with the same price cannot have different sizes. As a consequence, the design team has newly gathered the semantically meaningful FD Item, Price → Size.

So far, however, the literature has mainly looked at Armstrong relations. In this paper, we will investigate structural and computational properties of Armstrong bags for the combined class of keys and functional dependencies. We will show how to transfer results known for functional dependencies over relations (Beeri et al. 1984, Mannila & Räihä 1986) to characterize and compute Armstrong bags for keys and functional dependencies. More precisely, using the notions of agree and maximal attribute sets we will characterize when a given bag is an Armstrong bag for a given set of keys and functional dependencies. Using this structural characterization, we will then establish an algorithm that computes an Armstrong bag for a given set of keys and functional dependencies. While the time-complexity of computing Armstrong bags is precisely exponential in the size of the given constraints in general, our algorithm computes an Armstrong bag whose number of rows is at most quadratic in the minimum number of rows required. Armstrong bags can be created from a given set of constraints. Vice versa, we may ask for a given bag for which set of constraints this bag is Armstrong. This is the problem of constraint discovery from data, and as it turns out, it is also very useful in the context of constraint acquisition.

Example 3 Coming back to Example 2, the domain experts may have legacy data available, or may want to modify the data provided by the design team. For example, they provide the following bag to the design team:

<table>
<thead>
<tr>
<th>Item</th>
<th>Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>takoyaki</td>
<td>large</td>
<td>¥90</td>
</tr>
<tr>
<td>takoyaki</td>
<td>large</td>
<td>¥90</td>
</tr>
<tr>
<td>nigiri sake</td>
<td>large</td>
<td>¥90</td>
</tr>
<tr>
<td>nigiri sake</td>
<td>average</td>
<td>¥70</td>
</tr>
<tr>
<td>nigiri toro</td>
<td>small</td>
<td>¥70</td>
</tr>
<tr>
<td>okonomiyaki</td>
<td>small</td>
<td>¥300</td>
</tr>
</tbody>
</table>

The designers would now welcome tools to discover the set of keys and functional dependencies satisfied by this bag. It turns out that a cover for this set consists of no keys and the two FDs Item, Size → Price and Item, Price → Size.

In our next contribution we show how to compute a cover for the set of keys and FDs that hold in a given bag. This allows the database designer in our running example, to discover the set of keys and FDs that hold in the bag of Example 3. The problem of dependency discovery is not only important in database design, but also in database re-organization and for query optimization, among others (Mannila & Räihä 1994). We can combine the schema-driven approach (computing Armstrong bags) with the sample-driven approach (discovering dependencies) to the discovery of keys and functional dependencies. That is, we compute small semantic samples of existing bags. A semantic sample of a given bag is a bag contained in the given bag that satisfies the same keys and functional dependencies. The size, however, of the semantic sample is usually much smaller than the size of the given bag. In the literature such semantic samples are also known as informative Armstrong samples (De Marchi & Petit 2007). The benefit of informative Armstrong samples is that they consist of real-world tuples. This is in contrast to Armstrong samples computed from a given constraint set only. In the latter, a design team would need to substitute artificial values by real-world domain values first, before presenting the sample to the domain experts. Yet, the combination of real-world domain values may still not contain a real-world tuple. Therefore, informative Armstrong samples take advantage of the additional input in form of legacy data.

1.3 Organization

We comment on related work in Section 2. The data model, including the framework of keys and FDs is defined in Section 3. We characterize the structure of Armstrong bags in Section 4, and also show how to compute Armstrong bags with a small number of tuples. The problem of dependency discovery for keys and FDs in bags is analyzed in Section 5. We conclude and comment on future work in Section 6.

2 Related Work

Data dependencies have been studied thoroughly in various data models, and for the purpose of this paper it is not useful to aim at a complete overview. Mainly, we will focus on work related to the results established in this paper, i.e., to the following areas concerning keys and FDs: axiomatisations and implication problem, Armstrong samples, dependency discovery and informative Armstrong samples.

Keys and FDs are concepts almost as old as the relational model of data itself (Codd 1970). Armstrong established the first axiomatization of FDs under set semantics (Armstrong 1974), now known as the Armstrong axioms. In fact, Armstrong showed that the Armstrong axioms are even strongly complete for the implication of FDs, i.e., for an arbitrary relation schema and an arbitrary set of FDs on that schema, he constructed a single finite set of tuples which satisfies precisely all implied FDs. That is the reason, why such specific relations became known as Armstrong relations and more generally as Armstrong databases for more general classes of data dependencies.

Armstrong databases constitute an invaluable tool for the validation of semantic knowledge, and a user-
friendly representation of integrity constraints. Armstrong relations have been deeply studied for keys (DeMiguel 1980, Thalheim 1989) and FDs (Armstrong 1974, Beeri et al. 1984, Demetrovics et al. 1998, Mannila & Räihä 1986). They have also been analyzed for various other classes of data dependencies. An excellent survey on Armstrong databases is (Fagin 1982a), and a brief review of recent results is (Link 2012). In (Hartmann et al. 2012) Armstrong databases for the combined class of keys and functional dependencies were over partial bags were studied. Here, null values are permitted to occur in columns that are null-able. The class of keys and FDs over bags are a special case where no attribute is null-able. However, we establish a construction of Armstrong bags which generally requires only half the number of tuples as the Armstrong samples constructed in the special case of total bags in (Hartmann et al. 2012). That is, the special case of total bags enjoys an optimization in terms of the size of the Armstrong sample constructed, and we establish this optimization here.

The concept of informative Armstrong databases was introduced (De Marchi & Petit 2007). These are small sets of keys and functional dependencies for keys and FDs over bags. They have also been analyzed for various other classes of data dependencies.

We will adapt the hypergraph transversal approach to the discovery of keys and FDs in bags. The characterizations of sets of keys and FDs satisfied by a relation in terms of hypergraph transversals was also observed in (Thi 1986). The problem of dependency discovery has also been studied for other data dependencies, e.g., multivalued dependencies, inclusion dependencies, excluded dependencies, embedded dependencies, branching and fractional dependencies, and FDs in XML Data dependency discovery is an attractive problem in machine learning (Flach & Savnik 1999): the inference of general rules from instances of data. The problem is attractive since there always exists a set of dependencies that fits the instance exactly, and therefore dependency discovery can be solved exactly, while in inductive learning there is always a possibility of error (Mannila & Räihä 1994). Recently, the approximation variant of the dependency discovery problem has been studied (Giannella & Robertson 2004, Huhntala et al. 1999). In this setting, various measures for the error of a dependency in a relation are considered. These error measures have the value 0 if the dependency holds and a value close to 1 if the dependency clearly does not hold. To the best of our knowledge, dependency discovery has not been studied over bags yet.

3 Preliminaries

In this section we will define the underlying concepts of our study. These include the definition of schemata, relations and bags, but also keys and FDs. Subsequently, we will briefly summarize the notions of and some results on the implication and inferences for keys and FDs.

3.1 Schemata and Instances

A bag schema is a finite, non-empty set $B$ of attributes. Every attribute $A$ of a bag schema $B$ is associated with a domain $dom(A)$, an at most countably infinite set of possible values of $X$. A tuple over $B$ is a function $t : B \rightarrow \bigcup_{A \in B} dom(A)$ such that for all $A \in B$ we have that $t(A) \in dom(A)$. For a subset $X \subseteq B$ the projection of a tuple $t$ over $B$ on $X$ is the restriction $t|X$ of $t$ to $X$. For subsets $X$ and $Y$ of $B$ we write $XY$ for the set union $X \cup Y$. If $X = \{A_1, \ldots, A_n\}$, then we may write $A_1 \cdot \cdot \cdot A_n$ for $X$. In particular, we may write simply $A$ to denote the singleton $\{A\}$. A bag over $B$ is a finite multiset of tuples over $B$, usually denoted by $b$. We will use $\{\{\}\}$ to enumerate the elements of a bag explicitly, e.g. $\{\{t, t\}\}$ consists of two occurrences of the tuple $t$. A relation over $B$, usually denoted by $r$, is a finite set of tuples over $B$. In the context of a relation, we usually speak of a relation schema, usually denoted by $R$, rather than a bag schema. Commonly, we will speak of a schema $S$ if we refer either to a relation schema or to a bag schema. Furthermore, we will speak of an instance if we refer either to a relation or to a bag. Instances can be illustrated as tables with each tuple of the instance corresponding to a row of the table. The attributes of the corresponding schema may be used as column headers.

Example 4 Consider the bag schema ORDER with attributes Item, Size and Price from Example 1. We may choose the same domain STRING for all the attributes. Quite naturally, duplicate tuples occur in bags over ORDER: some customer's order might consist of 3 large takoyaki for the price of ¥90 each:

<table>
<thead>
<tr>
<th>Item</th>
<th>Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>takoyaki</td>
<td>large</td>
<td>¥90</td>
</tr>
<tr>
<td>takoyaki</td>
<td>large</td>
<td>¥90</td>
</tr>
<tr>
<td>takoyaki</td>
<td>large</td>
<td>¥90</td>
</tr>
</tbody>
</table>

For example, the projection of the tuple (takoyaki, large, ¥90) on $\{\text{Item}, \text{Price}\}$ is (takoyaki, ¥90).

Integrity constraints are specified as semantic constraints on schemata. They enable us to model real-world instances of a schema by restricting the set of possible instances to those considered meaningful to the application at hand. One of the most fundamental classes of integrity constraints are keys and FDs. Keys enable us to uniquely identify tuples within an instance.

Definition 1 Let $B$ be a bag schema. A key over $B$ is an expression $kX$ where $X$ is a non-empty subset of $B$. A bag $b$ over $B$ is said to satisfy the key $kX$ over $B$, denoted by $b \models kX$, if every pair of distinct tuples in $b$ deviates on some attribute in $X$, i.e., for all $t_1, t_2 \in b$ we have: if $t_1 \neq t_2$, then there is some $A \in X$ such that $t_1(A) \neq t_2(A)$ holds. In other words, $b$ satisfies the key $kX$ if for every $t_1, t_2 \in b$ the following holds: if $t_1(X) = t_2(X)$, then $t_1 = t_2$.

While any relation over the relation schema $R$ satisfies the key $kR$, proper bags (those that are not a set) do not satisfy any key over the associated bag schema.

Example 5 Consider the following bag $b$ over ORDER:

<table>
<thead>
<tr>
<th>Item</th>
<th>Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>okonomiyaki</td>
<td>average</td>
<td>¥150</td>
</tr>
<tr>
<td>okonomiyaki</td>
<td>average</td>
<td>¥150</td>
</tr>
</tbody>
</table>

None of the subsets $X$ of ORDER=$\{\text{Item}, \text{Size}, \text{Price}\}$ satisfies the property that for all distinct $t_1, t_2 \in b$ we have $t_1(X) \neq t_2(X)$.

In relational databases, i.e., when we consider relations over relation schemata, then the concept of a key can be generalized by the concept of a functional dependency.
**Definition 2** Let $B$ denote a bag schema. A functional dependency (FD) over $B$ is an expression $X \rightarrow Y$ where $X, Y \subseteq B$. A bag $b$ over $B$ is said to satisfy the FD $X \rightarrow Y$ over $B$, denoted by $b \models X \rightarrow Y$, if for every pair of tuples in $b$ that match on all attributes in $X$ also match on all attributes in $Y$, i.e., for all $t_1, t_2 \in b$ we have: if $t_1(X) = t_2(X)$, then $t_1(Y) = t_2(Y)$ holds.

**Example 6** Consider again the bag schema ORDER with attributes Item, Size and Price. Intuitively, the same item of the same size should always have the same price. This “business rule” can be formalized as the functional dependency

$$\text{Item, Size} \rightarrow \text{Price}. $$

Notice that the bags in Examples 4 and 5 satisfy this functional dependency. Since each instance is expected to satisfy this functional dependency, instances such as

<table>
<thead>
<tr>
<th>Item</th>
<th>Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>okonomiyaki average</td>
<td>450</td>
<td>Y450</td>
</tr>
<tr>
<td>okonomiyaki average</td>
<td>500</td>
<td>Y500</td>
</tr>
</tbody>
</table>

are prohibited from entering the database. Notice that it does not make sense to specify the functional dependency

$$\text{Item} \rightarrow \text{Price}$$

since there might be same items that have different prices (e.g. because they are different in size), for example

<table>
<thead>
<tr>
<th>Item</th>
<th>Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item,Size</td>
<td>average</td>
<td>Y450</td>
</tr>
<tr>
<td>Item,Size</td>
<td>small</td>
<td>Y300</td>
</tr>
</tbody>
</table>

The challenge for the database designer is to identify all keys and FDs that are meaningful to the application domain.

**Example 7** Note that the bags in Examples 4 and 5 satisfy every FD over ORDER, but violate every key over ORDER. No set of constraints that consists exclusively of FDs ever implied any key over bags.

### 3.2 Implication and inference

For the design of a schema, constraints are commonly specified as semantic constraints on the intended instances of the schema. During the design process one needs to determine further constraints which are implied by the given ones. In the following, $C$ denotes a class of integrity constraints, say the combined class of keys and FDs.

**Definition 3** Let $S$ denote a schema, and let $\Sigma \cup \{\varphi\}$ be a set of integrity constraints of class $C$ over $S$. We say that $\Sigma$ implies $\varphi$, denoted by $\Sigma \models \varphi$, if and only if each instance over $S$ that satisfies all $\sigma \in \Sigma$ also satisfies $\varphi$.

In order to determine the logical consequences of a set of constraints one can utilize a syntactic approach by applying inference rules, e.g. those in Table 1. These inference rules have the form

| premise | conclusion |

and inference rules without any premises are called axioms.

Let $\Sigma \cup \{\varphi\}$ be a set of integrity constraints from a class $C$, all defined over a schema $S$. Furthermore, we use $\Theta$ to denote a set of inference rules. A finite sequence $\gamma = [\varphi_1, \ldots, \varphi_n]$ of integrity constraints from $C$ is called an inference from $\Sigma$ by $\Theta$ if and only if each $\varphi_i$ is either an element of $\Sigma$ or is obtained by applying one of the rules of $\Theta$ to appropriate elements of $\{\varphi_1, \ldots, \varphi_{i-1}\}$. We say that the inference $\gamma$ infers $\varphi_n$, i.e. the last element of the sequence $\gamma$, and write $\Sigma \vdash_{\Theta} \varphi_n$. Let $\Sigma_{\Theta} = \{\varphi \mid \Sigma \vdash_{\Theta} \varphi\}$ denote the syntactic closure of $\Sigma$ under inferences by $\Theta$. An inference rule is called sound if the set of constraints in the premise of the rule implies the dependency in the conclusion. The set $\Theta$ is called sound for the implication of constraints in $C$ if for every schema $S$ and for every set $\Sigma$ of constraints in $C$ over $S$ we have $\Sigma_{\Theta} \subseteq \Sigma^*$ if and only if $\varphi$ is sound. The set $\Theta$ is called complete for the implication of constraints in $C$ if for every schema $S$ and for every set $\Sigma$ of constraints in $C$ over $S$ we have $\Sigma^* \subseteq \Sigma_{\Theta}$. The (finite) set $\Theta$ is called a (finite) axiomatization for the implication of constraints in $C$ if it is both sound and complete for the implication of constraints in $C$. The rules of Table 1 form finite axiomatizations for the implication of keys, and of FDs, respectively, over relations (Armstrong 1974). The rules of Table 2 form a finite axiomatization for the implication of keys and FDs over bags (Köhler & Link 2010).

### 3.3 Interactions in relations and bags

Let us fix a relation schema $R$. A key $kX$ over $R$ implies the functional dependency $X \rightarrow R$ since in every relation over $R$ that satisfies the key $kX$ there can never be two distinct tuples that have matching values on all attributes in $X$. Vice versa, the functional dependency $X \rightarrow R$ also implies the key $kX$ over $R$: in every relation over $R$ no two distinct tuples can have matching values on all attributes in $R$, i.e., they must have non-matching values on some attribute in $X$ by means of the functional dependency $X \rightarrow R$. Hence, a key $kX$ is equivalent to the functional dependency $X \rightarrow R$ over the relation schema $R$, in the sense that they are satisfied by the same relations $r$ over $R$. Consequently, FDs subsume the concept of keys in the context of relational databases.

Let us now fix a bag schema $B$. A key $kX$ over $B$ still implies FDs $X \rightarrow R$ over $B$. However, the reverse direction does not hold: take the bag $b = \{(t, t)\}$ where $t$ denotes some tuple over $B$. The same bag $b$ shows that there does not necessarily need to be any key over $B$ that is satisfied by a bag. This situation is exemplified by Example 6. The functional dependency $\text{Item,Size} \rightarrow \text{Price}$ does not imply the key $k\{\text{Item,Size}\}$ on the bag schema $\{\text{Item,Size, Price}\}$.

### 4 Armstrong Bags

In this section we will investigate structural and computational properties of Armstrong bags for sets of keys and functional dependencies. Starting from the definition of an Armstrong bag we show first that any set of keys and functional dependencies has an Armstrong bag. Using the essential notions of agree sets and maximal attribute sets, we then establish sufficient and necessary conditions for a given set of keys and FDs. Based on these conditions we then establish an algorithm that computes an Armstrong bag for any given set of keys and functional dependencies. The number of tuples in the computed Armstrong bag is guaranteed
to be at most quadratic in the minimum number of tuples required by an Armstrong bag. The results extend the structural and computational properties of Armstrong relations known for sets of functional dependencies over relations (Beeri et al. 1984, Mannila & Räihä 1986). They also simplify the structural and computational properties of Armstrong samples known for sets of keys and functional dependencies over partial bags (Hartmann et al. 2012). In particular, the construction described in this paper required about half the number of tuples as the construction from (Hartmann et al. 2012) for partial bags applied to the special case of (total) bags.\\n\\n4.1 Existence of Armstrong Bags\\n\\nArmstrong samples are concise “user-friendly” representations of sets of data dependencies, they have important applications in the acquisition and validation of these dependencies (Beeri et al. 1984, De Marchi & Petit 2007, Fagin 1982a, Link 2012, Mannila & Räihä 1986). We begin with the definition of Armstrong bags.

**Definition 4** Let $C$ denote a class of constraints. A bag $b$ over bag schema $B$ is said to be a $C$-Armstrong bag for a given set $\Sigma$ of constraints in $C$ over $B$ if and only if for every constraint $\varphi \in C \subseteq B$ the following holds: $b$ satisfies $\varphi$ if and only if $\varphi$ implies $\varphi$. The class $C$ is said to enjoy Armstrong bags if and only if for every bag schema $B$ and every set $\Sigma$ of constraints in $C$ over $B$ there is a bag $b$ over $B$ that is $C$-Armstrong for $\Sigma$.

The finite axiomatization $\mathcal{F}$ from Table 2 for the implication of keys and FDs over bags was established in (Köhler & Link 2010). We use the construction in the completeness proof to establish the fact that keys and FDs enjoy Armstrong bags. Note that such a result should not be taken for granted. The class of functional and inclusion dependencies over relations, for example, does not enjoy Armstrong databases; and neither does the class of general keys and functional dependencies over partial bags (Hartmann et al. 2012). We assume that the domain of every attribute of any given bag schema has sufficiently many domain values.

Let $B$ be a bag schema, and $\Sigma$ a set of keys and FDs defined on $B$. For every key and every functional dependency $\varphi$ such that $\varphi \notin \Sigma^+$, we construct a bag $b_{\Sigma,\varphi}$ precisely as in the proof of Theorem 7 in (Köhler & Link 2010).

Let $b$ denote the union of the $b_{\Sigma,\varphi}$ for all $\varphi \notin \Sigma^+$. However, for every attribute $A \in B$, and for every element $d \in dom(A)$ we require that $d$ occurs in a two-element bag $b_{\Sigma,\varphi}$ for at most one $\varphi$. Note that this construction is possible since the domain of every attribute is assumed to have sufficiently many elements. However, for each attribute $A \in B$, that is, for each attribute $A$ such that $\theta \mapsto A \in \Sigma^+$, we require that for all $t, t' \in b$ we have $t(A) = t'(A)$, i.e., every $A$ entry in every tuple in $b$ is the same value. It follows that $b$ satisfies the keys and FDs in $\Sigma$, and violates every key and every FD over $B$ not implied by $\Sigma$. Therefore, we obtain the following result.

**Theorem 1** Keys and FDs enjoy Armstrong bags.

### 4.2 Structural Characterization

Armstrong bags for a given set $\Sigma$ of keys and FDs must violate every FD $XY \rightarrow A$ not implied by $\Sigma$. Whenever $XY \rightarrow A$ is violated, then so is $X \rightarrow A$. Therefore, it suffices to violate the FD $XY \rightarrow A$, where $X$ is maximal with the property that $X \rightarrow A$ is not implied by $\Sigma$. For an attribute $A \in B$ let therefore

$$\text{max}(A) = \{ X | X \subseteq B, X \rightarrow A \notin \Sigma^+ \}$$

\[ \forall Y \subseteq B (X \subseteq Y \Rightarrow (Y \rightarrow A \notin \Sigma^+)) \]

denote the set of all maximal attribute subsets of $B$ on which $A$ is not functionally dependent (Mannila & Räihä 1986). Furthermore, let $\text{MAX}(B) = \bigcup_{A \in B} \text{max}(A)$ and $\text{CL}(B) = \{ X | X \subseteq B, X_{\Sigma^+} = X \}$ (Mannila & Räihä 1986). Here, $X_{\Sigma^+} = \{ A \in B | X \rightarrow A \notin \Sigma^+ \}$ denotes the attribute closure of $X$ with respect to $\Sigma$. It follows that $\text{MAX}(B) \subseteq \text{CL}(B)$, and $B \in \text{CL}(B)$. It is known that $\text{MAX}(B) = \text{GEN}(B)$ (Mannila & Räihä 1986) where $\text{GEN}(B)$ denotes the unique minimal subfamily of generators in $\text{CL}(B)$ such that each element of $\text{CL}(B)$ can be expressed as an intersection of sets in $\text{GEN}(B)$ ($B$ is the intersection of an empty collection of sets).

For some bag $b$ over $B$ let $t, t' \in b$. The agree set of $t, t'$ (Mannila & Räihä 1986) is defined as $\text{ag}(t, t') = \{ A \in B | t(A) = t'(A) \}$. Moreover, the agree set of $b$ (Mannila & Räihä 1986) is defined as

$$\text{ag}(b) = \{ \text{ag}(t, t') | t, t' \in b, t \neq t' \}.$$

For relations the following result is known (Beeri et al. 1984, Theorem 6.1). Let $R$ denote a relation schema, and $\Sigma$ a set of FDs over $R$. Then for every relation $r$ over $R$ we have: $r$ is an Armstrong relation with respect to $\Sigma$ if and only if $\text{MAX}(R) \subseteq \text{ag}(\text{r}) \subseteq \text{CL}(R)$.

Note that for every bag schema $B$ and every bag $b$ over $B$ we have $B \subseteq \text{ag}(b)$ precisely if $b$ is not a relation. This indicates that the attribute set $B$ itself is required to characterize Armstrong bags, as illustrated by the following example.

**Example 8** Consider the bag schema $\text{Order}$ and the set $\Sigma$ that contains the functional dependency $\text{Item} \rightarrow \text{Price}$. The relation

<table>
<thead>
<tr>
<th>Item</th>
<th>Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>takoyaki</td>
<td>large</td>
<td>¥90</td>
</tr>
<tr>
<td>nigiri sake</td>
<td>large</td>
<td>¥90</td>
</tr>
<tr>
<td>nigiri sake</td>
<td>average</td>
<td>¥70</td>
</tr>
<tr>
<td>nigiri sake</td>
<td>small</td>
<td>¥70</td>
</tr>
<tr>
<td>okonomiyaki</td>
<td>small</td>
<td>¥300</td>
</tr>
</tbody>
</table>

satisfies the condition $\text{MAX}(B) \subseteq \text{ag}(b) \subseteq \text{CL}(B)$, but it is not an Armstrong bag with respect to $\Sigma$. For instance, it satisfies the key $k[\text{Item,Size}]$ which is not implied by $\Sigma$.

An Armstrong bag for a set $\Sigma$ of keys and functional dependencies must also violate all keys not implied by $\Sigma$. Suppose $kX$ is not implied by $\Sigma$ over bag schema $B$. If $X \rightarrow B$ is not implied by $\Sigma$, then $X \rightarrow A$ is not implied by $\Sigma$ for some $A \in B - X$. Hence, $X \subseteq M \in \text{max}(A)$. In this case, any bag that violates the FD $M \rightarrow A$ will also violate the key $kX$, including any Armstrong bag for $\Sigma$. Consider now the case where $X \rightarrow A$ is implied by $\Sigma$. Since $kX$ is not implied by $\Sigma$ it follows by the soundness of the pullback rule that the weakest possible key $kB$ is also not implied by $\Sigma$. This, however, means that $\Sigma$ does not contain any key (otherwise $kB$ would be implied by $\Sigma$).

We will now extend the structural characterization of Armstrong relations for FDs (Beeri et al. 1984, Theorem 6.1).

**Theorem 2** Let \( b \) be a bag over \( \mathcal{B} \), and let \( \Sigma = \Sigma_k \cup \Sigma_f \) be a set of keys and FDs over \( \mathcal{B} \). Then \( b \) is an Armstrong bag for \( \Sigma \) if and only if both of the following conditions are satisfied: i) \( MAX(\mathcal{B}) \subseteq ag(b) \subseteq CL(\mathcal{B}) \), and ii) \( \mathcal{B} \in ag(b) \) if and only if \( \Sigma_k = \emptyset \).

We now give the proof of Theorem 2. For that purpose we distinguish between two different cases. Let \( \Sigma_k \neq \emptyset \). If \( b \) is an Armstrong bag for \( \Sigma \), then it is a relation as it satisfies some key. Due to the characterization of Armstrong relations established in (Beeri et al. 1984, Theorem 6.1), condition i) is satisfied. Condition ii) is also satisfied since a relation cannot have duplicate tuples. If \( b \) satisfies conditions i) and ii), then it is a relation. Due to the characterization of Armstrong relations established in (Beeri et al. 1984, Theorem 6.1), \( b \) must be an Armstrong relation for \( \Sigma \).

It remains to consider the case where \( \Sigma_k = \emptyset \). Assume first that \( b \) satisfies conditions i) and ii). Due to condition ii), \( b \) violates every possible key over \( \mathcal{B} \). Condition i) ensures that \( b \) satisfies precisely those FDs implied by \( \Sigma_f \), cf. (Beeri et al. 1984, Theorem 6.1). Assume now that \( b \) is an Armstrong bag for \( \Sigma \). Since \( \Sigma_k = \emptyset \) it follows that \( b \) violates every possible key over \( \mathcal{B} \), in particular \( \mathcal{B} \) itself. Consequently, condition ii) is satisfied. Condition i) follows from the fact that \( b \) satisfies precisely those FDs implied by \( \Sigma_f \), cf. (Beeri et al. 1984, Theorem 6.1).

### 4.3 Computational Properties

Given a first approximation of the target data dependencies in form of a set \( \Sigma \) of explicitly specified keys and FDs on a bag schema \( \mathcal{B} \), a design team may want to validate this approximation with domain experts or users of the target database. These domain experts and users usually do not have the background to understand the meaning of \( \Sigma \). Instead, the design team may let a computer generate an Armstrong bag with respect to \( \Sigma \), substitute the artificial key for the one presented for the case of relations (Mannila & Räihä 1986). The following algorithm is an extension to understand the meaning of \( \Sigma \). Instead, the main experts and users usually do not have the background to understand the meaning of \( \Sigma \). Instead, the design team may let a computer generate an Armstrong bag for \( \Sigma \), substitute the artificial key for the one presented for the case of relations (Mannila & Räihä 1986).

**Algorithm 1** Armstrong Bag Computation

1. procedure ARMSTRONG-BAG(\( \mathcal{B}, \Sigma_k, \Sigma_f \))
2. compute \( MAX(\mathcal{B}) \):
3. for all \( A \in T \) do
4. \( t_0(A) \leftarrow c_{A,0} \)
5. end for
6. if \( \Sigma_k = \emptyset \) then
7. \( b \leftarrow \{ \{t_0, t_0\} \} \)
8. else
9. \( b \leftarrow \{ t_0 \} \)
10. end if
11. \( i \leftarrow 1 \)
12. for all \( W \in MAX(\mathcal{B}) \) do
13. for all \( A \in \mathcal{B} \) do
14. if \( A \in W \) then
15. \( t_i(A) \leftarrow t_{i-1}(A) \)
16. else
17. \( t_i(A) \leftarrow c_{A,i} \)
18. end if
19. end for
20. \( b \leftarrow b \cup \{ t_i \} \)
21. \( i \leftarrow i + 1 \)
22. end for
23. return \( b \)
24. end procedure

If \( \Sigma_k \neq \emptyset \), then we are in the case of relations. There, line 8 ensures that no duplicate tuples will occur in the initialization of \( b \). The remaining steps are the exact steps of the algorithm in (Mannila & Räihä 1986).

If \( \Sigma_k = \emptyset \), then Theorem 2 tells us that \( b \) must contain distinct tuples with agree set \( \mathcal{B} \). Therefore, line 6 ensures that such a duplicate tuple will occur in the initialization of \( b \). The remaining steps ensure that Algorithm 1 produces an Armstrong bag that satisfies precisely all the FDs implied by \( \Sigma_f \) (Mannila & Räihä 1986).

The time complexity of finding an Armstrong bag, given a set of keys and FDs, is precisely exponential in the size of the given constraints. That is, (1) there is an algorithm for obtaining an Armstrong bag, given the set \( \Sigma \) of keys and FDs, where the running time of the algorithm is exponential in the size of the constraints; and (2) there is a set \( \Sigma \) of keys and FDs in which the number of tuples in each Armstrong bag for \( \Sigma \) is exponential. The result is a consequence of (Beeri et al. 1984, Theorem 7.1).

**Theorem 4** The complexity of finding an Armstrong bag with respect to a given set of keys and FDs over a bag schema \( \mathcal{B} \) is precisely exponential in the size of the given keys and FDs.

Statement (2) above follows immediately from the construction in (Beeri et al. 1984, Theorem 7.1) since every relation is a bag. Statement (1) above follows also immediately from the construction in (Beeri et al. 1984, Theorem 7.1) in the case where \( \Sigma_k \neq \emptyset \). For the remaining case where \( \Sigma_k = \emptyset \) we simply pick a tuple \( t \) from the Armstrong relation \( r \) constructed in (Beeri et al. 1984, Theorem 7.1) and add another tuple \( t' \) to \( r \) which coincides with \( t \) on all attributes. It follows that \( b := r \cup \{ (t') \} \) is an Armstrong bag for \( \Sigma \).

Note that this is a worst-case analysis. There are also sets of keys and FDs where the number of tuples in a minimum-sized Armstrong bag is logarithmic in the size of an optimal cover of the constraint set. Consequently, there is no best representation of constraints, i.e., either in form of a constraint set or in form of an Armstrong sample. Indeed, it is best
to have both representations. Armstrong bags reveal semantically meaningful constraints perceived semantically meaningless, and representations in form of constraint sets reveal semantically meaningless constraints perceived semantically meaningful.

5 Dependency discovery

In Example 3 the guests and owners of our restaurant example have produced another bag as their “model” of data that they think is typical for the application domain at hand. The design team is faced with the problem of extracting those keys and FDs that hold in this bag. Indeed, the problem the design team faces is known as the problem of dependency discovery.

In our case, the team would like an algorithm that computes a cover of the set of keys and FDs that hold in the given bag. This is the reverse problem to computing an Armstrong bag given a set of keys and FDs. In this section, we will show how an algorithm for functional dependency discovery under set semantics can be adapted to key and functional dependency discovery under bag semantics. This algorithm will make extensive use of hypergraph transversals (Eiter & Gottlob 1995, Mannila & Räihä 1994). We argue that even in the presence of relations the discovery of FDs should include a discovery of all minimal keys, followed by the discovery of the remaining FDs not implied by the minimal keys.

5.1 Hardness of Dependency Discovery

We start this section by introducing some notation. Let \( \mathbf{b} \) denote a bag over bag schema \( \mathfrak{B} \). The set of all keys and FDs holding in \( \mathbf{b} \) is denoted by \( \text{dep}(\mathbf{b}) = \text{dep}_k(\mathbf{b}) \cup \text{dep}_f(\mathbf{b}) \), i.e., \( X \in \text{dep}(\mathbf{b}) \) if and only if \( \exists \mathbf{b} \in \mathbf{b} \) \( kX \), and \( X \rightarrow Y \in \text{dep}(\mathbf{b}) \) if and only if \( \exists \mathbf{b} \in \mathbf{b} \) \( kX \rightarrow Y \) over \( \mathfrak{B} \). If \( \Sigma \) and \( \Sigma' \) are equivalent sets of keys and FDs, i.e., all the dependencies of \( \Sigma \) imply those of \( \Sigma' \) and vice versa, then we say that \( \Sigma \) is a cover of \( \Sigma' \) (and \( \Sigma' \) is a cover of \( \Sigma \)). In general, \( \text{dep}(\mathbf{b}) \) has several covers of varying size. We now adapt some of the results on the complexity of dependency discovery to bags. The first result shows that for some bags over \( n \) attributes all the covers of their dependency sets are of exponential size in \( n \).

**Theorem 5** For each \( n \) there is a bag \( \mathbf{b} \) over \( \mathfrak{B} \) such that \( |\mathfrak{B}| = n, |\mathbf{b}| = \Theta(n^2) \), and each cover of \( \text{dep}(\mathbf{b}) \) has \( \Omega(2^{n^2/2}) \) dependencies.

Since every relation is a bag the corollary follows from the same statement that was established previously for relations (Mannila & Räihä 1992).

In practice, bags that have inherently large covers should be rare. Corollary 5 shows that the results of dependency discovery can be large. One can also show that it is hard to identify these dependency sets.

**Theorem 6** The problem of determining whether a bag has a key containing at most \( k \) attributes is NP-complete.

The problem is clearly in NP. The corresponding problem for relations has been shown to be NP-hard by a reduction from the **Vertex Cover Problem** which is NP-complete (Beeri et al. 1984). The same reduction applies here as every relation is a bag.

Note that the dependency discovery problem for keys in relations (and therefore bags) is also inherently exponential in the number of attributes. Even though the dependency discovery problems are inherently exponential, Mannila and Räihä developed a useful and practical algorithm for inferring FDs from relations. This algorithm has demonstrated a satisfactory efficiency when being applied to “real-life” databases (Mannila & Räihä 1994). We will now discuss the use of this algorithm in bags, and also discuss how we can discover useful covers directly.

5.2 Minimal Left-hand Sides & Transversals

A simple reduction of the key and functional dependency discovery problem for bags to the functional dependency discovery problem in sets works as follows: (1) we scan the input bag \( \mathbf{b} \) for duplicate tuples, (2) if such duplicate tuples exist, then \( \emptyset \) is a cover of \( \text{dep}_k(\mathbf{b}) \), else \( \{\mathbf{b}\} \) is a cover of \( \text{dep}_k(\mathbf{b}) \), and (3) we compute a cover of \( \text{dep}_f(\mathbf{b}) \) by applying any algorithm for functional dependency discovery to the input \( \mathbf{b} \), e.g. those in (Mannila & Räihä 1994).

However, the set \( \text{min-key}(\mathbf{b}) = \{X \in \mathfrak{B} \mid \mathbf{b} X \land \neg \exists Y \subset X (\neg(Y \subset X)) \} \) of all minimal keys that hold in \( \mathbf{b} \) is a more desirable cover of \( \text{dep}_k(\mathbf{b}) \) that has considerable advantages for database design, maintenance, query optimization and database tuning. The computation of \( \text{min-key}(\mathbf{b}) \) would also not increase the exponential time complexity of computing a cover of \( \text{dep}_f(\mathbf{b}) \). Hence, we aim to find a cover of \( \text{dep}_k(\mathbf{b}) \) that contains \( \text{min-key}(\mathbf{b}) \), and where the set of remaining FDs (those that hold in \( \mathbf{b} \) but are not implied by \( \text{min-key}(\mathbf{b}) \) is small. Alternatively to this strategy, one may compute all minimal keys from the cover obtained by the simple reduction above. However, unless \( P=NP \), it takes exponential time to compute a minimal key, given a set of FDs. Thus, we prefer to discover all minimal keys directly from the input bag.

To keep things consistent with previous work (Mannila & Räihä 1994) we only consider standard FDs in this section, i.e., FDs \( X \rightarrow A \) where \( X \subset \emptyset \) and \( A \subset \mathfrak{B} \). Given \( \mathfrak{B} \), and given an attribute \( A \subset \mathfrak{B} \), denote by \( \text{lhs}(A) \) (left-hand sides of \( A \)) the set of minimal nontrivial attribute sets \( X \subset \mathfrak{B} \) such that \( X \rightarrow A \) is implied by \( \Sigma \). That is, \( \text{lhs}(A) = \{X \subset \mathfrak{B} \mid \Sigma \models X \rightarrow A \land \forall Y \subset X (\Sigma \models Y \rightarrow A)\} \). We borrow some material from hypergraphs (Eiter & Gottlob 1995). A collection \( \mathfrak{H} \) of subsets of \( \mathfrak{B} \) is a (simple) hypergraph, if no element of \( \mathfrak{H} \) is empty and if \( X, Y \in \mathfrak{H} \) and \( X \subset Y \) imply \( X = Y \). The elements of \( \mathfrak{H} \) are called edges of the hypergraph, and the elements of \( \mathfrak{B} \) are called the vertices of the hypergraph.

We first note that \( \text{lhs}(A) \) is a hypergraph.

Given a simple hypergraph \( \mathfrak{H} \) on \( \mathfrak{B} \), a transversal \( T \) of \( \mathfrak{H} \) is a subset of \( \mathfrak{B} \) intersecting all the edges of \( \mathfrak{H} \), that is, \( T \cap E \not= \emptyset \) for all \( E \in \mathfrak{H} \). Transversals are also called hitting sets. A minimal transversal of \( \mathfrak{H} \) is a transversal such that no \( T' \subset T \) is a transversal. The collection of minimal transversals of \( \mathfrak{H} \) is denoted by \( \text{Tr}(\mathfrak{H}) \). It is a hypergraph on \( \mathfrak{B} \). The next algorithm computes \( \text{Tr}(\mathfrak{H}) \) for a simple hypergraph \( \mathfrak{H} \) (Mannila & Räihä 1994).

**Algorithm 2** Minimal Transversals of Hypergraphs

1. **procedure** \( \text{Transversals}(\mathfrak{H} = (V, E)) \)
2. \( \text{Tr}(\mathfrak{H}) \leftarrow \{\emptyset\} \)
3. for all \( E \in \mathfrak{H} \) do
4. \( \text{Tr}(\mathfrak{H}) \leftarrow \{X \cup \{e\} \mid X \in \text{Tr}(\mathfrak{H}) \land e \in E\} \)
5. \( \text{Tr}(\mathfrak{H}) \leftarrow \{X \in \text{Tr}(\mathfrak{H}) \mid \neg \exists Y \in \text{Tr}(\mathfrak{H}) (Y \subset X)\} \)
6. end for
7. return \( \text{Tr}(\mathfrak{H}) \)
8. **end procedure**
5.3 Dependency Discovery Algorithm

Let \( \text{disag}(b) = \{ (\mathcal{B} - \text{ag}(t, t') ) \mid t, t' \in b \land t \neq t' \} \) denote the complements of agree sets of distinct tuples. Furthermore, let \( \text{disag}(b) = \{ X \in \text{disag}(b) \mid \neg \exists Y \in \text{disag}(b) (Y \subset X) \} \) contain the minimal sets of \( \text{disag}(b) \). We then have that \( \text{min-key}(b) = Tr(\text{disag}(b)) \) (Thi 1986).

Let \( \text{disag}(b, A) = \{ (\mathcal{B} - \text{ag}(t, t')) \mid t, t' \in b \land t \neq t' \} \} \) denote the complements of agree sets of those tuples that do not match on the attribute \( A \). Note that \( \text{disag}(b, A) = \{ W \in \text{disag}(b) \mid A \in W \} \).

Furthermore, let \( \text{disag}(b, A) = \{ X \in \text{disag}(b, A) \mid \neg \exists Y \in \text{disag}(b, A) (Y \subset X) \} \) contain the minimal sets of \( \text{disag}(b, A) \). Then it is known that \( \text{lhs}(A) = Tr(\text{disag}(b, A)) \) (Mannila & Räihä 1994, Theorem 5).

Next we present the key and functional dependency discovery algorithm. Lines 2-8 compute the minimal keys that hold in the input bag \( b \), lines 9-14 produce a cover of the FDs that hold in \( b \), and lines 15-19 compute a non-redundant cover of the set of FDs in the presence of the minimal keys. Other covers might be computed instead (Maier 1980).

Algorithm 3 Dependency Discovery

1. procedure DISCOVERY(\( \mathcal{B}, b \))
2. \( J_b := \{ (\mathcal{B} - \text{ag}(t, t')) \mid t, t' \in b \land t \neq t' \} \}
3. if \( \emptyset \in J_b \) then
4. \( \Sigma_k = \emptyset \)
5. else
6. \( K_b := \{ W \in J_b \mid \neg \exists V \in J_b (V \subset W) \} \}
7. \( \Sigma_f = Tr(K_b) \)
8. end if
9. for all \( A \in \mathcal{B} \) do
10. \( J_A := \{ W \in J_b \mid A \in W \} \)
11. \( K_A := \{ W \in J_A \mid \neg \exists V \in J_A (V \subset W) \} \)
12. \( L_A := Tr(K_A) \)
13. end for
14. \( \Sigma_f := \{ X \rightarrow A \mid A \in \mathcal{B} \land X \in L_A \} \}
15. for all \( \sigma \in \Sigma_f \) do
16. if \( \Sigma_k \cup \Sigma_f - \{ \sigma \} = \sigma \) then
17. \( \Sigma_f \leftarrow \Sigma_f - \{ \sigma \} \)
18. end if
19. end for
20. return \( \Sigma_k \cup \Sigma_f \)
21. end procedure

Example 9 Let \( b \) denote the bag from Example 3 that is input to Algorithm 3. The set \( J_b \) contains the sets \( \emptyset \), \{Item\}, \{Item, Size, Price\}, \{Size, Price\}, \{Item, Size\}, and \{Item, Price\}. Consequently, line 7 returns \( \Sigma_k = \emptyset \). Line 11 generates the following K-families:

- \( K_{\text{Item}} = \{ \text{Item} \} \)
- \( K_{\text{Size}} = \{ \text{Size}, \text{Price}; \text{Item}, \text{Size} \} \)
- \( K_{\text{Price}} = \{ \text{Size}, \text{Price}; \text{Item}, \text{Price} \} \)

The L-families from line 12 are:

- \( L_{\text{Item}} = \{ \text{Item} \} \)
- \( L_{\text{Size}} = \{ \text{Size}; \text{Item}, \text{Price} \} \)
- \( L_{\text{Price}} = \{ \text{Price}; \text{Item}, \text{Size} \} \)

Line 14 produces the FDs \( \text{Item} \rightarrow \text{Item}; \text{Size} \rightarrow \text{Size}; \text{Item, Price} \rightarrow \text{Size}; \text{Price} \rightarrow \text{Price} \); and \( \text{Item, Size} \rightarrow \text{Price} \). Finally, lines 15-19 produce \( \Sigma_f = \{ \text{Item, Price} \rightarrow \text{Size}; \text{Item, Size} \rightarrow \text{Price} \} \).

Algorithm 3 compute directly all minimal keys that hold in \( b \). That is, there is no need to apply an exponential time algorithm to compute the set of minimal keys from a given set of FDs subsequently after the dependency discovery. We exemplify this by using an example from (Mannila & Räihä 1994).

Example 10 Let \( r \) denote the following relation

\[
\begin{array}{cccc}
E(employee) & D(department) & M(manager) & S(salary) \\
\text{Smith} & \text{Toys} & \text{Jones} & 200 \\
\text{Wilson} & \text{Admin} & \text{Brown} & 300 \\
\text{Barnes} & \text{Toys} & \text{Jones} & 300 \\
\end{array}
\]

Algorithm 3 produces the following output on input \( r \):
\( \Sigma_1 = \{ E, DS, MS \} \) and \( \Sigma_f = \{ D \rightarrow M, M \rightarrow D \} \). On the other hand, the Algorithm from (Mannila & Räihä 1994) would produce the following cover: \( \{ E \rightarrow D, D \rightarrow M, E \rightarrow S, D \rightarrow M, M \rightarrow D, DS \rightarrow E \} \).

Example 10 illustrates another application area of dependency discovery. While query optimization can benefit from the data dependencies enforced on the schema, dependency discovery can also infer additional dependencies that hold accidentally in the current instance. This provides further useful information to optimize queries.

Theorem 7 Algorithm 3 computes a cover for dep(\( b \)) in time polynomial in \( |b|, |\mathcal{B}|, \prod_{W \in \text{disag}(b)} |W| \) and the \( \prod_{W \in \text{disag}(b, A)} |W| \).

The size of the \( J_A \)-family is bounded by that of \( J_b \). The size of \( J_b \) is \( \mathcal{O}(b^2) \). The \( K_A \)-family is a subcollection of the \( J_A \)-family, and can be found in time polynomial in \( |b| \) and \( |\mathcal{B}| \). Computing the sets in \( Tr(K_b) \) and the \( L_A \) can be done in time polynomial with respect to \( \prod_{W \in \text{disag}(b)} |W| \) and \( \prod_{W \in \text{disag}(b, A)} |W| \) respectively, by using Algorithm 2.

6 Conclusion and Future Work

Duplicate detection has evolved into its own field of research (Naumann & Herschel 2010). In bags dependencies interact differently than they do in relations. As duplicates occur in real database systems, it is desirable to extend existing results from relations to bags. We have established a schema-driven and a data-driven approach to the discovery of keys and FDs in bags. For the schema-driven approach we compute Armstrong bags from a given set of keys and FDs. Armstrong bags visualize and communicate current perceptions of the domain semantics between stakeholders of the target database. This can lead to the discovery of semantically meaningful constraints previously perceived meaningless. For the data-driven approach we mine given bags for the set of keys and FDs it satisfies. These algorithms can be combined to generate informative Armstrong bags from a given bag, and serve as small semantic data samples to visualize the semantics currently present in a database.

For future work it would be interesting to develop database normalization algorithms that apply to bags, e.g., for 3NF, BCNF, or 4NF decompositions (Arenas & Libkin 2005, Biskup et al. 1979, Ferrarotti et al. 2012, Kolahi 2007, Vincent 1999). It is not clear what a lossless decomposition of a bag constitutes. The performance of Algorithm 3 can be improved (Mannila & Räihä 1994), and different types
of FD discovery algorithms should be adapted to the setting of bags. See (Lin et al. 2012) for a recent survey on dependency discovery algorithms.

References


Fagin, R. (1982a), Armstrong databases, Technical Report RJ3440(0926), IBM Research Laboratory, San Jose, California, USA.


Improving Remote Collaborative Process Modelling using Embodiment in 3D Virtual Environments

Erik Poppe¹, Ross Brown¹, Jan Recker¹, Daniel Johnson¹

1 Queensland University of Technology, 126 Margaret Street, Brisbane QLD4000, Australia
{erik.poppe, r.brown, j.recker, dm.johnson}@qut.edu.au

Abstract
Identifying, modelling and documenting business processes usually require the collaboration of many stakeholders that may be spread across companies in inter-organizational settings. While modern process modelling technologies are starting to provide a number of features to support remote collaboration, they lack support for visual cues that are present in co-located collaboration. In this paper, we examine the importance of visual cues for collaboration tasks in collaborative process modelling from distributed remote locations. Based on this analysis, we present a prototype 3D virtual world process modelling tool that supports a number of visual cues to facilitate remote collaborative process model creation and validation. We report on a preliminary analysis of the technology and also describe the future direction of our research with regards to the theoretical contributions expected from the evaluation of the tool.

Keywords: Virtual Environments, Avatars, Collaboration, Business Process Modelling

1 Introduction
Business process modelling is a key practice in business process management (van der Aalst et al. 2003), which has been a top priority in many enterprises for a number of years now (Gartner Inc. 2010) as they invest in efforts to (re-) design organizational or technological systems. Process modelling is concerned with graphically describing the business processes of an organization (Indulska et al. 2009). To create process models, modelling experts have to extract and consolidate the domain knowledge that is distributed among all the people involved in the business process (Dean et al. 2000). This happens in the form of communication between the modelling expert(s) and the domain experts in group workshops or individual interviews (S. Hoppenbrouwers et al. 2005).

Tool support for process modelling has been shown to affect the perceptions (Recker 2012) of stakeholders as well as to increase the participation in process improvement projects (Kock 2001). This is especially so when domain experts are scattered across multiple locations in a large multinational company or in global projects, when technology is a required mechanism to facilitate communication and collaboration. Technology that supports synchronous communication such as audio or video conferencing is broadly available today. This technology, however, does not support a number of visual cues that are often used for efficient collaboration on artefacts.

In this paper, we report on research that specifically explores technology support for visual cues in synchronous communication to aid the process of collaborative process modelling in distributed settings. Based on this research we hypothesise that the use of avatars in a virtual environment will facilitate remote collaborative process model creation and validation, by providing visual cues that are critical for efficient collaboration. To that end, we report on the development of a prototype solution based on 3D virtual world technology and outline our research plan that will examine the impact of such communication features on collaborative model validation and correction tasks.

2 Background
Our research builds on, and integrates, three streams of literature: First, we need to understand how process modelling is conducted, and how this process changes when relevant stakeholders need to collaborate remotely. Second, we need to understand how technology can be used to facilitate communication in remote collaborative tasks. Finally, we need to specifically understand how visual cuing can be used to alleviate communication problems in remote collaborations. We discuss these issues, in turn.

2.1 The Process of Process Modelling
Process modelling transforms knowledge about the processes of a business into accurate models (Scholz-Reiter & Stickel 1996). These models are governed by process modelling grammars or languages, which provide a set of constructs and rules about how these constructs can be used to represent real-world phenomena (Wand & Weber 2002).

The knowledge required for this transformation is usually distributed across a range of people internal and sometimes external to a business. Each of these stakeholders has a mental model of the process. Through collaboration, these models are adjusted iteratively until every participant has the same mental model (S. Hoppenbrouwers et al. 2005). Accordingly, process modelling can be described as a process of converging on a shared view (J. Hoppenbrouwers et al. 2005) that is
These visual cues are especially relevant in remote coordination and a common ground in collaboration. They are used to monitor the status of the collaborative task, the participants’ actions and comprehension, to direct and identify the focus of attention and create efficient messages. These effects have been demonstrated in a number of experiments. Specifically, full gaze-awareness (Monk & Gale 2002), situated actions (Gergle et al. 2004) and gesturing (Dodds et al. 2011) have all been shown to positively impact communication.

In summary, when communicating, people can make use of visual cues provided by head, body and shared objects to provide efficient feedback. This facilitates coordination and a common ground in collaboration. These visual cues are especially relevant in remote collaboration tasks involving the design and use of artefacts. We now review how visual cues relate to the process of process modelling specifically.

2.3 Visual Cues in Collaborative Process Modelling

Co-located process modelling is often conducted in workshops where several stakeholders are situated around a process model in form of a print-out or projection. The physical co-presence in the workshop setting facilitates the process of grounding communication between participants because the shared visual space supports a variety of visual cues. For example, the bodies of every workshop participant improve situational awareness by providing an indication of a) who is present, b) who is working together with whom and c) who is working on which part of the model. They therefore support awareness of the part of the process currently being discussed. Fig. 1 provides an example.

![Fig. 1: Awareness and spatial context in collaborative process modelling.](image)

The body can also be used for non-verbal communication (Fig. 2). Through gesturing and gaze the discussion can be regulated, structured and illustrated. For example, nodding to demonstrate agreement or understanding can

<table>
<thead>
<tr>
<th>Participants Heads and Faces</th>
<th>Participants Bodies and Actions</th>
<th>Task Objects</th>
<th>Work Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 - Facial expression can be used to identify how close to agreement the team is at any stage.</td>
<td>A2 - Inferences about intended changes to task objects can be made from body position and actions.</td>
<td>A3 - Changes to task objects can be directly observed</td>
<td>A4 - Activities and objects in the environment that may affect task status can be observed</td>
</tr>
<tr>
<td>B1 - Gaze direction can be used to infer intended actions</td>
<td>B2 - Body position and actions can be directly observed</td>
<td>B3 - Changes to task objects can be used to infer what others have done</td>
<td>B4 - Traces of others’ actions may be present in the environment</td>
</tr>
<tr>
<td>C1 - Eye-gaze and head position can be used to establish others’ general area of attention</td>
<td>C2 - Body position and activities can be used to establish others’ general area of attention</td>
<td>C3 - Constrains possible foci of attention</td>
<td>C4 - Constrains possible foci of attention; disambiguates off-task attention (e.g. disruptions)</td>
</tr>
<tr>
<td>D1 - Gaze can be used as a pointing gesture</td>
<td>D2 - Gestures can be used to illustrate and refer to task objects.</td>
<td>D3 - Pronouns can be used to refer to visually shared task objects</td>
<td>D4 - Environment can help constrain domain of conversation.</td>
</tr>
<tr>
<td>E1 - Facial expressions and nonverbal behaviors can be used to infer level of comprehension</td>
<td>E2 - Appropriateness of actions can be used to infer comprehension and clarify misunderstandings</td>
<td>E3 - Appropriateness of actions can be used to infer comprehension and clarify misunderstandings</td>
<td>E4 - Appropriateness of actions can be used to infer comprehension and clarify misunderstandings</td>
</tr>
</tbody>
</table>

Table 1: Visual cues for coordination and common ground. Adapted from Kraut et al. (2003)
provide efficient feedback to the speaker without interrupting. Pointing directs a joint focus of attention and therefore allows for efficient referencing of model parts and locations.

**Fig. 2: Synchronous verbal and non-verbal communication in collaborative process model modelling**

Furthermore, some non-verbal cues, such as facial expressions and body posture, can be without communicative intent of participants, as shown in Fig. 3. For example, gazing away indicates a decline in interest, which in turn impacts consensus-building and group decision-making processes in relation to the modelling artefact.

**Fig. 3: Unintentional non-verbal communication in collaborative process modelling**

In summary, communication in process modelling tasks can be facilitated by a number of visual cues when a visual space is shared by participants, for example due to physical co-presence. We now examine how much support is offered by current technology in remote settings where physical co-presence is limited.

### 2.4 Tool Support for Collaborative Process Modelling

When co-presence in a shared physical space is not possible or not efficient, communication needs to be supported by technology in order to allow for collaboration.

While verbal communication can be easily supported by phones or VOIP applications, shared visual features and spaces are more difficult to support. Video conferencing supports some visual features, such as facial expression and body posture, but literature shows that it has several limitations (Gaver 1992). The limited and stationary view into the other person’s immediate environment makes it hard to see at what location the person is looking or pointing. Hindmarsh (2000) concludes that most synchronous communication systems fail to accommodate the processes people use to establish mutual orientation when collaborating on artefacts at a co-located site. Process modelling technologies have started to address the problem of collaboration in process modelling. Commonly used generic drawing tools such as Microsoft Visio do not support collaboration explicitly. Users have to save models to files and send these via email, separate repositories or other collaborative systems. BizAgi (2012) provides a locking mechanism to signal individual edits to a model, but no functionality to share models. Signavio (2009) solves this problem by storing models in a centralized repository. However, Signavio does not provide support for synchronous communication. Hahn et al. (2010) found that there is little support for synchronous collaboration in most process modelling tools they examined. Mendling et al. (2012) similarly observed that some aspects of awareness and communication are poorly supported by current process modelling tools. More recently, however, tools have started to implement synchronous collaboration features such as chat functions and synchronous model viewing and editing. IBM Blueworks Live (IBM 2010) and SAP StreamWork (SAP 2010) provide synchronous communication tools in the form of text chat. ARIS Business Architect and ProcessWave support audio and video chat.

These tools, therefore, provide support for the use of shared artefacts for coordination and common ground, but not for any of the cues that require an explicit embodiment in the modelling space. To support this argument, we have studied a focussed collection of tools, as well as any available documentation and publications relating to them, to analyse the functional support of any of the visual cues listed in Table 1. Table 2 shows the visual cues supported by each of the tools mentioned above, compared to the proposed prototype tool. Cues are listed as ‘fully supported’ when the behaviour described for the cues can be reproduced with the software. They are marked as ‘partially supported’ when meaningful parts of this behaviour can be reproduced using the tool. Otherwise, they are marked as ‘not supported’. As can be seen, visual cues that use the body and actions of participants are not supported by any of the tools. Even though some tools show an image of the participant, it is never used in relation to the process model. We now discuss technology that can support these kinds of visual cues.
applications in the past, a rapid increase of the processing (Herring & Borner 2003). While the intense computation and the use of face-to-face communication patterns (Ott & Dillenbourg 2002), allow avatars. Embodiment in virtual environments reduces environments is to use 3D virtual environments with other visual applications in society today, and so the skills required for such environments should not be a significant impediment anymore. Desktop-based virtual environments, however, still have a number of limitations associated with visual interface technology. Keyboard and mouse controls do not limit the users expressiveness when using the avatar (Mazalek et al. 2011) and do not support unintentional non-verbal communication, such as facial expression or body posture. Furthermore, the use of a monitor as a window into the virtual world results in a limited field-of-view that can make it difficult to see both the origin and target of a pointing gesture (Hindmarsh et al. 1998). Reviews of technology supporting remote collaboration (Otto et al. 2006; Wolff et al. 2007) suggest that immersive interfaces can overcome these limitations. Recently, Dodds et al (2011) demonstrated that using a virtual world with fully animated avatars increased the speed of convergence to a mutual understanding between two people in a word guessing game. Traditionally, technology used for immersive interfaces was not feasible on a consumer level, however, more recently more cost-effective availability of large stereoscopic displays, head-mounted displays and motion sensing input devices have made them attainable to consumers even with a limited budget. In summary, collaborative virtual environments with immersive interfaces can support visual features that are not well supported by other technologies.

2.5 Collaborative Virtual Environments

![Desktop-based virtual environments](image)

Fig. 4: Desktop-based virtual environments

One way to support visual features in collaborative work environments is to use 3D virtual environments with avatars. Embodiment in virtual environments reduces referential ambiguity (Ott & Dillenbourg 2002), allows the use of deixis (referencing something through context, e.g. “the window here”), pointing (Hindmarsh et al. 2000) and the use of face-to-face communication patterns (Herring & Borner 2003). While the intense computation required to display virtual worlds has limited their applications in the past, a rapid increase of the processing performance of devices has enabled the use of technologically advanced virtual environments, even on smart phones (Epic Games 2010). Similarly, the additional skills required by users to navigate virtual environments effectively, are to date mostly existent in significant share of people due to the popularity of video games (Entertainment Software Association 2012) and other visual applications in society today, and so the skills required for such environments should not be a significant impediment anymore. Desktop-based virtual environments, however, still have a number of limitations.

### Table 2: Visual Cues supported by tool (✓ – fully supported, (✓) – partially supported, ✗ – not supported)

<table>
<thead>
<tr>
<th>Tool vs Feature</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Visio</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1</td>
</tr>
<tr>
<td>BizAgI</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2</td>
</tr>
<tr>
<td>Signavio</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2</td>
</tr>
<tr>
<td>IBM Blueworks</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>4</td>
</tr>
<tr>
<td>SAP StreamWork</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>4</td>
</tr>
<tr>
<td>ARIS Business Architect</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>6</td>
</tr>
<tr>
<td>ProcessWave</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>6</td>
</tr>
<tr>
<td>West et. AI Prototype</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>9</td>
</tr>
<tr>
<td>Research prototype</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>14</td>
</tr>
</tbody>
</table>

In our previous work we have explored the use of 3D virtual worlds to support collaborative process modelling (West et al. 2010; Brown 2010). We have shown how the SecondLife platform can be used for collaborative process modelling. Participants responded positively to the practice of collaborative process modelling.
In addition, the modelling environment developed in SecondLife had a number of technical limitations (Poppe et al. 2012). The usage of SecondLife required the use of scripts and specific chat commands to interact with the process modelling tool. The fixed grid layout of the process model made it difficult to add elements, since the limited space often required moving other model elements out of the way first.

Furthermore, while the SecondLife client can be modified, modifications that require interaction with the server usually require complex workarounds. An example of such functionality is the use of motion capture to animate avatars. Since SecondLife was developed to use predefined animations it does not synchronise the skeletons used for character animation over the network. This makes addition of such features unnecessarily complex.

It was therefore decided to develop a dedicated prototype tool instead, in which model representation and interactions, as well as the user interface, can be designed specifically for the task of process modelling.

4 Prototype Process Modelling Tool

We developed a prototype business process modelling tool (Fig. 5) to support collaborative process modelling with avatars in a 3D virtual environment.

4.1 Implementation

One major requirement guided the development: since we want to investigate the effect of visual cues in process modelling we wanted the tool to be similar to current process modelling tools in all other regards. For this reason we use the BPMN grammar for the process model. We have implemented 64 process model elements from the BPMN standard, including swim lanes, all activities, events, gateways and three types of sequence flow.

We also decided to represent the process model in a 2D plane, so that users can interact with the model in the same way they would normally interact with process models in present 2D tools. Having all model elements on a flat plane also ensures that there are fewer issues with occlusion caused by the overlying of objects viewed from a particular point of view, which can be a problem for 3D data representations (Tominski et al. 2005). Furthermore, we implemented a graphical drag & drop interface (see Fig. 6) similar to commonly used modelling tools such as the Signavio Process Editor (Signavio 2009).

Fig. 5: Collaborative process modelling tool prototype

![Collaborative process modelling tool prototype](image)

Fig. 6: Illustration of the Drag & Drop user interface

Users can create model elements by dragging the image of the required element from a bar at the top of the screen into the 3D space. They can move and scale elements by dragging markers on their corners (see Fig. 7). Even though the process model is two dimensional it is placed in a three dimensional virtual environment and the user can look at it from different angles. From some of these...
viewing angles, text can be difficult to read. We therefore implemented floating labels that turn towards the camera, so that they are always visible to the reader and an algorithm that can move them within local boundaries to minimize occlusion.

Since the tool is primarily built to support collaborative process modelling, it provides a number of features for collaboration. First of all, it allows users to host a server or connect to a server. This server synchronizes all actions between the different clients. Connected users can then create, view and edit process models in a shared virtual space. All participants can see these changes in real-time to allow for communication and coordination by actions.

For purposes of communication the tool supports voice-over-IP (VOIP) and text chat. Furthermore, each user is embodied in this space with an avatar, therefore allowing referential shortcuts such as “the gateway over here”.

We have also implemented functionality to animate the avatars to support visual cues that depend on gesturing. This is used in three ways. First, avatars are automatically animated while the user interacts with the tool, e.g. a typing animation is played while the user enters text. Users can also choose specific predefined animations such as a head nod or waving an arm from a menu. Finally, we provide a procedural pointing animation, which users can execute by clicking on a model element while holding the Ctrl-key. This makes the avatar point at a selected element.

Furthermore, a command history gathers all the changes made to the model and changes can be undone by any of the participants. The history also contains an “awareness” display that shows what participants are currently doing (e.g. “User X is typing”) to allow for better coordination of both communication and editing.

Due to the nature of the process modelling task, we have also implemented two consensus mechanisms (see Fig. 8). The process model can be locked for validation. In this mode changes to the model cannot be done until every participant has marked a model element as error. Thereby, participants have to reach consensus before editing the model. Once a model element has been marked as an error, changes to the element can be applied. These changes are gathered in a changes list per element. Before these changes are made persistent each participant has to approve the list of changes. This feature requires participants to come to an agreement that the proposed solution is an appropriate solution to the observed modelling problem. Note that the consensus mechanism also allows different solutions to a modelling error to be accepted, if all participants agree on the proposed solution.

Since the virtual environment is not bound by the physics of the real world, people can edit the model from any distance and can teleport instantly to locations to minimize time spent traversing the 3D space.

In the final phase we will use the Microsoft Kinect as an immersive interface to automatically capture the body posture and motion as well as the facial expression of the user and display them in real-time on the avatar.

The prototype process modelling tool described provides the required functionality for remotely located users to collaboratively model business processes. In addition, the representation of users via avatars enables the use of visual cues for efficient communication. We now analyse in detail which visual cues are supported by each of the prototype’s features.
4.2 Feature analysis

In the following we show how the various features of the prototype tool described above translate into support for visual cues that facilitate collaborative process modelling as per Table 1.

The first group of cues rely on the use of shared task objects in communication. The prototype tool supports these by synchronising changes to the process model in real-time between all participants. This provides support for a number of visual cues:

- **A3 - Changes to task objects can be directly observed**
  Visible modifications to model elements allow participants to see whether somebody has implemented changes they discussed or is in the process of implementing them.

- **B3 - Changes to task objects can be used to infer what others have done**
  Visible modifications to model elements allow participants to infer that one person has implemented the discussed changes.

- **C3 - Constrain possible foci of attention**
  Since all the participants will be able to see the same elements, the focus of their discussion can be limited to these elements.

- **D3 - Pronouns can be used to refer to visually shared task objects**
  Participants can refer to model elements using pronouns. For example a participant might ask “Are you talking about this gateway?” moving the element back and forth to draw the attention of the other participants.

- **E3 - Appropriateness of actions can be used to infer comprehension and clarify misunderstandings**
  Participants can infer comprehension from changes made to model elements. For example, people can see someone has not understood which element they wanted to change as soon as the wrong element starts being moved.

Shared task objects are already supported by some of the current process modelling tools. However, a large number of visual cues require the use of an embodiment in a shared space with the process model. The prototype tool supports those by representing each user with an avatar in the modelling space. Specifically, the visual cues listed below are supported by the user embodiment in the space of the process model.

- **B1 - Gaze direction can be used to infer intended actions**
  The rotation of the avatar and avatar head shows which model elements are in the view of each user and thus can be interacted with.

- **C1 - Eye-gaze and head position can be used to establish others’ general area of attention**
  The rotation of avatars and avatar heads can be used to infer the current centre of attention for each participant.

- **A2 - Inferences about intended changes to task objects can be made from body position and actions.**
  The position of an avatar can show whether a participant is about to make the changes requested to the diagram.

- **B2 - Body position and actions can be directly observed**
  The position of the avatar can serve as an anchor in the diagram.

- **C2 - Body position and activities can be used to establish others’ general area of attention**
  The position of the avatars floating over the diagram can show which part of the model a participant is currently looking at.

- **D2 - Gestures can be used to illustrate and refer to task objects.**
  Because a user is embodied in the space of the diagram he can use deixis for efficient communication of references. He can say “Come over here, I found a problem in the model.” and other users will understand the meaning attached to the word “here” based on the position of his avatar.

- **E2 - Appropriateness of actions can be used to infer comprehension and clarify misunderstandings**
  A user can monitor whether the other participants understood him, by monitoring whether their avatars move towards his position as requested or look around the diagram trying to identify the target location.

Additional visual cues are supported via animation of the body. Automatically animating the body depending on the actions the user is performing on the process model improves awareness and support for the following visual cue:

- **B2 - Body position and actions can be directly observed**
  Animations on the avatar can show current activities of the user, e.g. a typing animation of an avatar that is hovering above a specific task in the model can show that the user is currently changing the label of the task.

Enabling users to animate the avatar and point at model elements with key-presses enables the use of non-verbal communication behaviours and improve support for the following visual cue:

- **D2 - Gestures can be used to illustrate and refer to task objects.**
  The users will be able to use gestures to communicate. Animations on the avatar can be used for pointing gestures since they are in one continuous space with the diagram and other users can see both the gesturing of the avatar as well as the relation of the gesture to the model or other participants.

The deliberate use of non-verbal communication, however, does not provide collaborators with information that is communicated unintentionally via body posture.
and facial expressions. The use of the Microsoft Kinect as an immersive interface to automatically animate the avatars enables further visual cues that allow for such communication:

- **A1 - Facial expression can be used to identify how close to agreement the team is**
  The automatic animation of the avatars face will make it easier for the users to see how many participants are in agreement.

- **E1 - Facial expressions and nonverbal behaviours can be used to infer level of comprehension**
  The automatic animation of the avatars face will give other users the opportunity to infer the state of a user and react to it, for example to clarify a point just made, when a user looks confused.

- **D2 - Gestures can be used to illustrate and refer to task objects.**
  Animations on the avatar can be used for illustrating gestures and other users can see both the gesturing of the avatar as well as the relation of the gesture to the model or other participants. The automatic animation should make timing of back-channel feedback, such as head nods, much more effective as the user does not require time to select an animation anymore.

- **E2 - Appropriateness of actions can be used to infer comprehension and clarify misunderstandings**
  The body posture of the avatar can be used to infer confusion.

As can be seen, by the end of the final phase, the tool will provide support for all head and body visual cues that have been identified by Kraut et al. (2003). In turn, the tool will extend the range of current process modelling tools and research prototypes indicated in Table 2 with sets of extended visual cue support. However, definitive evidence of the benefits of these features for collaborative process modelling has to be gathered empirically. We, therefore, describe an experimental design to evaluate the prototype tool in the next section.

### 5 Outlook: Empirical Evaluation

For an empirical evaluation of this prototype tool we will measure the impact of visual cues on key tasks in the collaborative process modelling process. In order to isolate this effect we will use an experimental setting that allows for control over other influences. As our primary interest, at this stage, is in the impact of embodiment, we plan to compare the performance of the prototype with and without this feature (as opposed to comparing the prototype to other process modelling tools). In order to evaluate the prototype tool we will use an experimental between-groups design to measure changes in team performance brought about by the addition of visual cues. This approach should ensure high internal validity, although at the expense of ecological validity.

For the experiments, we will use business process modelling students as proxies for novice process modellers. We will have groups of three process modellers use the prototype tool remotely and collaboratively for two specific process modelling tasks, that is, model validation and model correction. The process model chosen is that of a human digestive process, because we can assume this process to be reasonably difficult for participants to understand, in turn increasing the need for communication and collaboration. We have modified an expert validated base model to add 3 syntactic and 3 semantic errors for the experiments. The diagram consists of 164 elements. This makes purely individual search difficult, the complexity of reference high, and therefore should emphasise the effects we expect to find (Gergle et al. 2004). Furthermore, most participants in our study will have incomplete knowledge of this process, which we believe will motivate collaboration with other participants to reduce uncertainty.

The groups of participants will be asked to find and correct the errors in the diagram. We will assign each group to either the “with avatars” or the “without avatars” condition and will measure the number of errors found and fixed by the group, as well as the time taken to do so. This will then give us a performance measure to compare the average performance of groups using avatars and groups not using avatars.

To avoid any confounding influences brought into the experiment by the participants, prior modelling knowledge, domain knowledge and experience with virtual environments of each participant are measured using a questionnaire.

In addition we will measure factors that are a result of our experiment setup. The novelty of the tool may influence the participants’ enjoyment and affect their performance. We will therefore use the cognitive absorption measure used by Agarwal and Karahanna (Agarwal & Karahanna 2000) to measure such an influence. Furthermore, the 2D representation of the process model in a 3D space could make the model so difficult to read that the task performance of the users is affected. We will therefore use an ease of model interpretation measure from Burton-Jones and Meso (Burton-Jones & Meso 2008) to measure whether the ease of understanding the model representation affects the performance of the participants. Finally, we will measure whether participants found the experimental task difficult or the tool difficult to use and how that affected their performance. We will measure this using a subjective cognitive load measure (Paas et al. 2003).

The execution of this research design will allow us to test whether adding embodiments to support visual cues in a shared visual modelling space decreases the time required for collaborative process modelling.

### 6 Limitations

Within the scope of this research, it is difficult to fully replicate the feature sets of 2D professional tools. Therefore any comparison with a 2D tool developed by the team would be biased by definition, as the developed 2D modelling tool will be inferior to use, and so confound the experimental results in the comparison. We therefore focus in this project on developing best practice approaches and tools for 3D virtual world modelling systems alone.

A limitation of this study resulting from this is the ecological validity of the experiments. We are not
experimentally comparing the prototype tool to currently available process modelling tools and we are not evaluating them with professional process modellers. We can therefore not measure at this point in time, whether an increased overhead from having to navigate a 3D space as opposed to a 2D diagram would nullify any benefits gained from the additional visual cues.

7 Conclusions and Future work
In this paper we have shown how visual cues facilitate collaborative process modelling. We have found that current process modelling tools do not support these visual cues well and have identified technologies that can support them. We have furthermore proposed a modelling tool that uses these technologies to support these visual cues and have presented the first version of this tool. We then presented a research design to test the impact support for visual cues has on collaborative process modelling.

In the future, additional experimentation should evaluate the prototype in direct comparison with current process modelling tools, to supplement the high internal validity of our current investigation with greater ecological validity. However, as stated, this will require a mature, well-tested 3D tool to be developed first.

8 Acknowledgements
This research has been funded by the Smart Services CRC in Australia -http://www.smartservicescrc.com.au.

9 References


Using Formal Concept Analysis for Ontology Maintenance in Human Resource Recruitment

Dominic Looser\textsuperscript{1}  Hui Ma\textsuperscript{2}  Klaus-Dieter Schewe\textsuperscript{1}

\textsuperscript{1} Software Competence Center Hagenberg, Austria
\textsuperscript{2} Victoria University of Wellington, School of Engineering and Computer Science, Wellington, New Zealand

dominic.looser@tsch.at/kdschewe@acm.org

\textsuperscript{2} Hui.Ma@ecs.vuw.ac.nz

Abstract

Ontologies have been proven useful for many applications by enabling semantic search and reasoning. Human resource management has recently attracted interest by researchers and practitioners seeking to exploit ontologies for improving the efficiency and effectiveness of the job recruitment process. However, the quality of semantic search and decision making intimately depends on the quality of the ontology used. Most current efforts concentrate on the development of general ontologies that find wide approval by the HR community worldwide. In order to be useful for automatic matchmaking between job offers and job seekers, such high-level ontologies need to be adequately enriched with detailed domain-specific knowledge and adapted to the particular needs of individual job markets. We present an approach for enriching and adapting an existing ontology using formal concept analysis.

1 Introduction

Ontologies play an important role for semantic search and reasoning in many different important domains, including medical science, geographic information systems, and human resource recruitment applications. Ontology is used as knowledge base that represents domain knowledge that some work can be processed without or with little involvement of human domain experts. Human resource recruitment resource process is getting more and more time consuming due to the increasing number of applicants and jobs advertised on the Internet. To improve the process of human recruitment, researchers propose to use ontology-based semantic match approaches. For example, (Colucci, Noia, Sciscio, Donini, Mongiello & Motola 2003) proposes a formal approach to ontology-based semantic matching of skills descriptions, which is based on description logic formalization and reasoning. This approach can cope with cases where there is no exact match exist and provide matching results with a certain degree of matching. (Sánchez, Martínez-Béjar, Contreras, Fernández-Breis & Nieves 2006) presents an ontology-guided search engine to provide intelligent matches between job offers and CVs. Similarity matching has been used in web-based job applications for competence matching, e.g. (Bizer, Heese, Morchol, Oldakowski, Tolksdorf, Berlin & Eckstein 2005, Colucci et al. 2003, Fazel-Zarandi, Devlin, Huang & Contractor 2011, Herder 2009). Semantic matching for job recruitment relies on ontologies that capture domain knowledge of human resource recruitment experts. Therefore, it is crucial to construct a quality ontology as a knowledge base of the system. Building ontologies is very time consuming and involves many domain experts. The ontology hierarchy is the backbone of an ontology. In human resource domain some competence taxonomies, e.g. DISCO, O*Net, have been constructed so that it can be shared by many organisations. There is an increasing number of applications using the openly available ontologies to save the effort of building ontology from scratch. However, often the openly available standard ontologies need to be extended to include some specific domain knowledge of a particular organisation. Also, ontologies need to be maintained to reflect the change of the part the world it describes, in particular, when ontology is constructed and learned from data sets in various forms, e.g. documents or databases. Any changes to the data sets might require updates of the existing ontologies. For example, when human resource experts produce different matching results than an ontology-supported system one needs to analyse the differences and capture the new domain knowledge and use it to revise the existing ontology so that the matching results can be improved by using the revised ontology. If we manually revise the existing ontology, it is not only inefficient but also may cause inconsistencies. Therefore, it is desirable to develop an ontology maintenance approach so that an ontology can be improved in a consistent and systematic way. The goal of our research is to provide a method for revising existing ontologies for job recruit systems so that quality of ontologies can be ensured and semantic matching can be improved. Description logics are promising tools for managing ontologies because they provide not only formal representation models but also reasoning tools. For example, Chimaera (Lammari & Métais 2004) is based on description logics. (Lammari & Métais 2004) presents some techniques for building an Is_A ontology based on the description of ontology. This approach takes as input a set of instances and a set of constraints to generate concepts and the Is_A hierarchy. In the mean time there are some works using Formal Concept Analysis (FCA) to build ontologies. FCA was first introduced in (Wille 1982). In (Prediger & Stumme 1999), FCA is used to set up a so called Conceptual Information System when database is only partially given in the beginning. It shows how description logic and formal concept analysis can be connected and enriched. In particular, it shows how to use a subsumption algorithm of DL as an ‘expert’ for Attribute Exploration. However, it is not clear how DL is used in the process of using FCA. (Cimiano, Hotho & Staab 2005) presents a method of automatic acquisition of taxonomies or concept hierarchies from a text corpus based on FCA.
FCA is used to produce a lattice that is then converted into a special kind of partial order that constitutes a concept hierarchy. In (Jia, Newman & Tianfield 2007) FCA is applied to generate information context from a tentatively domain specific scientific corpus which is then mapped to a formal ontology. A semi-automatic method to ontology design is presented in (Haav 2004), with a set of rules mapping a FCA lattice to a rule language is presented. One of the advantages of using FCA for learning taxonomies is that it produces concepts that reflect the real life concept relations. To combine the advantages of DL and FCA, in this paper we will propose a FCA-based approach of maintaining ontology, which makes use of DL as modelling tool to model ontology of the knowledge base and use FCA to design and revise ontology.

The aim of our approach is to provide a framework for automatically updating and revising existing ontology hierarchies so that it can be used to provide better matching results in the process of human resource recruitment. In this paper we use the IT domain as an example to demonstrate our proposed approach. We will show that the revised ontology can indeed improve the quality of matching results. For that we use some existing approaches of semantic matching.

**Organisation.** The paper is organised as the following. In Section 2, we will present a motivating example. In Section 3 we will briefly review description logic (DL) and formal concept analysis (FCA), followed by a brief comparison between DL and FCA. In Section 4 we will present a framework that uses DL and FCA in the process of ontology maintenance. Section 5 will relate our work with existing approaches of using FCA and DL in the process of ontology design. Finally, Section 6 concludes the paper with a brief summary of our contribution.

## 2 Motivating Scenario

Job recruitment often involves processing a big number of applications for an open position. It is not efficient and effective to shortlist candidates manually. The internet has become more and more important platform for job recruitment for both job providers and job seekers. There are many big organisations setting up online application systems, where basic data of job profiles and application profiles are entered so that the data can be used for automated selection of candidates who have satisfied the competencies and other requirement in the job profiles. On the other hand job seekers prefer to use web portals to search for jobs that fits their competencies and other preferences. In both cases, it is a matter of matching between job demands and job offers.

When a job is advertised on the internet, a job profile will be created which specifies information about company name, location, industry, job title, job function, job type (full time or part time), length of contract, salary, demanded competencies/skills, etc. Personal profiles would typically include name, education, experience, preferred location, preferred job type, preferred salary range, competencies/skills. Among these we would need to match education, experience, location, salary and competencies/skills. The most difficult part for semantic matching is competencies.

Consider a job provider having a job position that requires candidates to have a set of programming skills, e.g. $D = \{\text{Java, ER, HTML}\}$. There are two candidates with one having programming skills $S_1 = \{\text{Java, UML, PHP}\}$ and the other candidates having programming skills $S_2 = \{\text{Java, ER, HTML}\}$. In the literature matching between job positions and job candidates involves the distance between the concepts on an ontology hierarchy. One can use some standard competence dictionary, e.g. IEEE Reusable Competency Definition (IEEE RCD 2008) and DISCO (Müller-Riedlhuber 2009).

DISCO (Müller-Riedlhuber 2009) uses a multilingual taxonomy of competencies to provide web users with a supporting tool for filling in and translating online forms. They can be used to build competency profiles for CVs or job offers using a unique vocabulary. However, high-level competency definitions are usually not detailed or complete enough to serve as an adequate knowledge base for competency management by HR or for job recruitment. The competencies defined in DISCO, for example, are too coarse-grained to be useful for precisely describing the skills that are available or needed. In consequence, reasoning tools will not identify any suitable candidate matching the skill requirement of the job. However, human resource experts may decide that the candidate $S_1$ will fit the job as he/she has the skill of using UML and should have the similar knowledge of ER because class diagram of UML can be used for modelling data. How are we going to refine the ontology hierarchy so that the system could select $S_1$ directly? In this paper we will present an approach of revising ontology using new domain knowledge.

### 3 Preliminaries

#### 3.1 Description Logics

Description logics (DL) are a family of logic languages for representing knowledge and reasoning about it (Baader, Calvanese, McGuinness, Nardi & Patel-Schneider 2003). The knowledge of interest for some applications is represented in terms of individuals, concepts and roles, and stored in a knowledge base. Concepts stand for sets of individuals, while roles stand for relationships to other individuals. A DL knowledge base is a set of axioms, and usually consists of two parts: a terminological knowledge layer (called TBox) and an assertional knowledge layer (called ABox). The TBox describes the terminology in use for the application, that is, defines concepts and states additional constraints on their interpretation. The ABox describes the individuals, that is, contains assertions that relate individuals to concepts.

We briefly review syntax and semantics of description logics. Let $N_C$ and $N_R$ be fixed sets of concept names and of roles names. One can then build complex concept expressions out of them by using the concept constructors provided by the particular description logic being used, see Table 1. Let $C$ denote the set of complex concept expressions that can be obtained by finitely many applications of these constructors. The members of $N_C$ are called atomic concepts. We further use the empty concept $\top$ as a shortcut for $\neg\bot$, and $(\leq m).R$ is a shortcut for $\neg(\geq m + 1).R$.

**Example 3.1** For our HR application we can use atomic concepts like $\text{SoftwareEngineer}$ or $\text{DBA}$, and roles like $\text{hasCompetency}$ or $\text{knows}$. Further, we can form concept expressions like $\text{hasQualification}\_\text{MBA}$ describing all individuals who have no other qualification than an MBA, or $\text{SoftwareEngineer} \sqcap \text{hasCompetency}\_\text{UML}$ describing all software engineers with UML skills.

Note that description logics correspond to fragments of first order logic: individuals correspond to constants, concepts to unary predicates, and roles to binary predicates. Due to restrictions set on their
expressiveness the associated decision problems like satisfiability are decidable. Different description logics vary by the concept constructors that they permit. The choice of a particular description logic is usually done by balancing expressiveness against the complexity of the associated decision problems. In this paper, we use $\mathcal{ALN}$ which is among the most popular description logics. The ideas discussed in this paper are general in the sense that they can be easily tailored to other reasonably expressive description logics. Recall that $\mathcal{ALN}$ is included in $\mathcal{SHOIN}^{(2)}$, the description logic underlying OWL-DL.

A subsumption axiom is a statement of the form $C_1 \subseteq C_2$ with concepts $C_1, C_2 \in C$. A terminology (or TBox) $\mathcal{T}$ is a finite set of subsumption axioms. We use the shortcut $C_1 \equiv C_2$ to denote both $C_1 \subseteq C_2$ and $C_2 \subseteq C_1$.

Interpretations are used to assign meaning to syntactic constructs. An interpretation $\mathcal{I}$ consists of a non-empty interpretation domain $\mathcal{O}$ and an interpretation function $\mathcal{I}(\cdot)$, which assigns to each atomic concept a subset of $\mathcal{O}$, and to each role a binary relation on $\mathcal{O}$. The interpretation $\mathcal{I}$ can be easily extended to concept expressions in $\mathcal{N}_C$. An interpretation $\mathcal{I}$ is a model of $\mathcal{T}$ if $\mathcal{I}(C_1) \subseteq \mathcal{I}(C_2)$ holds for every subsumption axiom $C_1 \subseteq C_2$ in $\mathcal{T}$. A model is finite if the interpretation domain $\mathcal{O}$ is finite. In this case, the model is also said to be an instance (or ABox) of $\mathcal{T}$.

A concept $C_1$ subsumes a concept $C_2$ if $\mathcal{I}(C_1) \subseteq \mathcal{I}(C_2)$ holds for every instance $\mathcal{I}$ of $\mathcal{T}$. We also write $\mathcal{T} \models C_1 \subseteq C_2$. We use $\mathcal{T} \models C_1 \equiv C_2$ as a shortcut for $\mathcal{T} \models C_1 \subseteq C_2$ and $\mathcal{T} \models C_2 \subseteq C_1$, and call $C_1, C_2$ equivalent.

Example 3.2 For our HR application we may use the subsumption axiom $\text{Software\_Engineer} \sqsubseteq \text{Has\_Competency\_Software\_Engineering}$ to state that software engineers must have a degree in software engineering or at least some software development skills.

The subsumption problem asks whether $\mathcal{T} \models C_1 \subseteq C_2$ holds for concepts $C_1, C_2$ and a TBox $\mathcal{T}$. The satisfiability problem asks whether $\mathcal{T} \models C \equiv \perp$ holds for a concept $C$ and a TBox $\mathcal{T}$. Both problems are decidable for description logics such as $\mathcal{ALN}$. For details, we refer to (Baader et al. 2003).

Example 3.3 For illustration we use a small fragment of the DISCO taxonomy (as shown in Figure 1) which gives rise to the following subsumption axioms in the TBox.

![Figure 1: Competency hierarchy](image)

3.2 Formal Concept Analysis

Next we review basic notions from FCA (Ganter & Wille 1999). A formal context is a triple $\mathcal{K} := (G, M, I)$, where $G$ is a set of individuals, $M$ is a set of properties (or attributes), and $I$ is a relation $I \subseteq G \times M$ that links each individual $a \in G$ to the properties satisfied by $a$. That is, $(a, b) \in I$ states that $a$ has property $b$. A formal concept is a pair $(A, B)$ such that $A$ and $B$ are maximal with $A \times B \subseteq I$. The set $A \subseteq G$ is called the extent and set $B \subseteq M$ is the intent of the concept. A formal concept $(A_1, B_1)$ is a subconcept of a formal concept $(A_2, B_2)$ if $A_1 \subseteq A_2$ (or equivalently $B_2 \subseteq B_1$). The concept lattice $\mathcal{L}(\mathcal{K})$ of $\mathcal{K}$ is the set of all its FCA concepts together with the subconcept/superconcept relation. As usual, we use lattice diagrams to illustrate the ordering relation among the formal concepts in a concept lattice. The nodes of the lattice are labelled by the respective FCA concepts. For the sake of clarity, however, node labels do not show the full extent and intent of an FCA concept, but show an individual $o$ only in the most specific FCA concept it belongs to.

Contexts can be represented as tables, with columns headed by the properties $b$ and rows headed by individuals $a$. The cells of the table are marked if and only if the relations holds for the corresponding pair of individual and property.

Example 3.4 Table 2 is a formal context with four object individuals and six properties.

<table>
<thead>
<tr>
<th>$a_1$</th>
<th>$b_1$</th>
<th>$b_2$</th>
<th>$b_3$</th>
<th>$b_4$</th>
<th>$b_5$</th>
<th>$b_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>$a_2$</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>$a_3$</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>$a_4$</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

Table 2: An example of formal context

Each of the objects owns some properties. For example, object $a_1$ holds properties $b_1, b_2, b_3$ and $b_6$ while object $a_3$ holds properties $b_1, b_3$ and $b_6$. If $A = \{a_1, a_3\}$, then $B = \{b_1, b_3, b_6\}$ is the intent of the formal concept $(A, B)$. That is, $A$ and $B$ is maximal with $A \times B \subseteq I$, i.e., properties $\{b_1, b_3, b_6\}$ is the maximum set of properties shared by the set of objects $\{a_1, a_3\}$ and $\{a_1, a_3\}$ is the maximum set of objects sharing the set of properties $\{b_1, b_3, b_6\}$.

Lets look at three formal concepts contained in the formal context, $(A_1, B_1)$, $(A_2, B_2)$, and $(A_3, B_3)$ with $A_1 = \{a_1, a_3, a_4\}$, $B_1 = \{b_1, b_6\}$, $A_2 = \{a_1, a_3\}$, $B_2 = \{b_1, b_3, b_6\}$, $A_3 = \{a_1\}$, $B_3 = \{b_1, b_3, b_4, b_5\}$. Concepts $(A_2, B_2)$ is a subconcept of concept $(A_1, B_1)$ and concept $(A_3, B_3)$ is a subset of concept $(A_2, B_2)$. The corresponding context lattice is shown in Figure 2.

Note that the subconcept/superconcept relation is transitive. For a superconcept $(A, B)$ all properties $b \in B$ are inherited by all its subconcepts.

The interplay between description logic and formal concept analysis has been addressed in the literature.
The partial order on hierarchy. Let C4

To build a concept lattice one should choose FCA properties (Sertkaya 2011). So when using FCA, we take an inputs an existing ontology hierarchy and output a revised ontology. To maintain and revise existing ontology hierarchies, we will show how FCA can be used to revise the ontology hierarchy of a DL.

We found it convenient to view a TBox as an ontology hierarchy. Let C1, . . . , Cn be the non-trivial concepts of interest for which subsumption axioms are recorded in a TBox T. Let M = {C1, . . . , Cn} and let ⊆ be the partial order on M induced by subsumption axioms in T (that is, the reflexive and transitive closure of T projected onto M). We call HH = (M, ⊆H) an ontology hierarchy of T. Figure 1 shows the diagram of an ontology hierarchy for the TBox in Example 3.3.

Ontologies need to be maintained to reflect the changing of the real world. For example GML emerged as a new standard for exchanging geographical data over the internet. This change should be reflected by the competency ontology so that the jobs requiring GML skills can be captured. Therefore, competency ontology should be revised when new knowledge is captured, e.g., information of new advised jobs or input of human resource experts. From information of new jobs we get some new objects that can be used to created FCA matrix. In this section we first present an ontology revision process. Then we will show how FCA can be used to revise the ontology hierarchy of a DL.

4.1 Ontology Revision Process

To maintain and revise existing ontology hierarchies, see (Sertkaya 2011) for a recent survey. While both study concepts, there are differences between the approaches. Most notable, DL and FCA use the term “concept” with different meanings, hence we will always speak of FCA concepts or DL concepts in the rest of the paper. FCA starts with the properties, and then inspects the complete extensional description of an application domain to conclude the concepts of this specific domain. DL, on the other hand, starts with an intensional definition of the concepts that is given independently of a specific domain (Baader & Sertkaya 2004). FCA uses atomic properties to classify individuals by the properties that they observe. DL uses atomic concepts and builds complex concept descriptions from them (and the roles). As pointed out by several authors, DL concepts correspond to FCA properties (Sertkaya 2011). So when using FCA to build a concept lattice one should choose M as the set NC of atomic DL concepts, and define I such that (a, b) ∈ I whenever individual a belongs to DL concept b in some ABox under inspection.

4 Revise Ontology Using FCA

We will show later in Section 4.2 that the representative context created provides the same implication as subsumption relations given by the ontology hierarchy. Also, we will show how to use the above process with some examples in the following subsections.

4.2 Generate Representative FCA Context

In the following let HH be a given ontology hierarchy, and M = {C1, . . . , Cn} the set of concepts defined in HH and ⊆H the subsumption relation of HH. We can then derive a representative ABox SHH for HH as follows: choose an n-element subset G = {g1, . . . , gn} of the interpretation domain O, and define the interpretation function I by I(C) = {gi ∈ G : C ⊑H C1} for C ∈ M.

It is easy to check that the interpretation I is indeed a model of HH. The ABox SHH can easily translated into a formal context KHH = (G, M, I) with (gi, C) ∈ I just when gi ∈ I(C) holds for the concept C ∈ M. In the FCA matrix of KHH we have a

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic concept</td>
<td>A</td>
<td>I(A) ⊆ O</td>
</tr>
<tr>
<td>role</td>
<td>R</td>
<td>I ⊆ O × O</td>
</tr>
<tr>
<td>universal concept</td>
<td>⊤</td>
<td>I(T) = O</td>
</tr>
<tr>
<td>negation</td>
<td>¬C</td>
<td>I(¬C) = O - I(C)</td>
</tr>
<tr>
<td>conjunction</td>
<td>C1 ⊓ C2</td>
<td>I(C1 ⊓ C2) = I(C1) ∩ I(C2)</td>
</tr>
<tr>
<td>disjunction</td>
<td>C1 ⊔ C2</td>
<td>I(C1 ⊔ C2) = I(C1) ∪ I(C2)</td>
</tr>
<tr>
<td>existential</td>
<td>∃R.C</td>
<td>I(∃R.C) = {x ∈ O</td>
</tr>
<tr>
<td>universal</td>
<td>∀R.C</td>
<td>I(∀R.C) = {x ∈ O</td>
</tr>
<tr>
<td>lower bound</td>
<td>(≥m.R)</td>
<td>I(≥ m.R) = {x ∈ O</td>
</tr>
</tbody>
</table>

Table 1: Syntax and semantics of $\mathcal{ALN}$.
row for every $g \in G$ such that there is a 1 in column $C$ if and only if $g$ satisfies concept $C$. We call $\mathbb{K}^H$ a representative FCA context for $\mathcal{H}$. Note that the properties of $\mathbb{K}^H$ are just the concepts of $\mathcal{H}$.

For $g_i \in G$ let $A_i = \{ g_i \}$ and $B_i = \{ C \in M : C_i \subseteq C \}$. It is easy to see that the pair $(A_i, B_i)$ forms an FCA concept of $\mathbb{K}^H$. We call it the FCA representative of the concept $C_i$. By definition, $(A_i, B_i)$ is a superconcept of $(A_j, B_j)$ just when $B_i \subseteq B_j$, and this holds if and only if $C_i \subseteq C_j$. We record this as follows:

**Proposition 1** The ontology hierarchy $\mathcal{H} = (M, \sqsubseteq_H)$ is isomorphic to the subconcept relation on the set of FCA concepts $(A_i, B_i)$ with $i = 1, \ldots, n$.

**Example 4.1** Following the process in 4.1, in Step 1 we create a representative ABox or a given ontology. For the example ontology hierarchy in Example 3.3 we associate an object for each concept in the hierarchy, say job descriptions $J_1, \ldots, J_{10}$, so that the following assertions are satisfied:

- Computing($J_1$)
- Database_Knowledge($J_2$)
- Database_Development($J_3$)
- Database_Modeling($J_4$)
- ER($J_5$)
- Software_Development($J_6$)
- Software_Design($J_7$)
- Software_Design_Method($J_8$)
- OO_Design($J_9$)
- UML($J_{10}$)

We can easily extend this to a representative ABox of the ontology hierarchy, and construct a representative FCA context for it, see Figure 4. Note that each job description $J_i$ holds the DL concept that it associates with and all the DL concepts above it on the ontology hierarchy. For example, $J_3$ holds the DL concept Database_Development and also the DL concepts above it, i.e. Database_Knowledge and Computing.

The corresponding FCA lattice is shown in Figure 5. The ontology hierarchy $\mathcal{H}$ given in Example 3.3 is isomorphic to the subconcept relation induced by the FCA representatives of the concepts in $\mathcal{H}$.

### 4.3 Update Ontology Hierarchies

In this section we present an algorithm for updating existing master ontology hierarchies. Our approach makes use of FCA contexts. Assume that the master ontology and the next generated context are from the same domain. There are some overlaps of attributes.

**Example 4.2** Consider the ontology in Example 4.1 and assume that a domain expert concludes that if a candidate knows UML then he/she has the skill of the database modelling because he/she knows Class Diagram. So a FCA context is created with one object, $J_{11}$, as shown in Figure 6.

According to the ontology revision process in Figure 3 we merge the two contexts, Master Skill Context and New Skill Context, into one as in Figure 7.

Note that when we are merging contexts we first check if there is any existing object sharing an ontology concept (or FCA attribute). For example we check if there is an object in the original matrix has the attribute UML. Because there is one we then merge the two objects into one $J_{10}$ and allocate all the attributes in $J_{10}$ and $J_{11}$ to it.

Using the updated context we can generate a FCA lattice as in Figure 8.

Note that if an object satisfies DL concept Database_Knowledge it also satisfies DL concept Computing because Database_Knowledge $\subseteq$ Computing.

As mentioned in Section 1, the motivation of maintaining ontology is to provide better matching results in the process of human resource recruitment. In the literature there are many approaches using ontology hierarchies to calculate distance or similarities between ontology concepts (Bizer et al. 2005, Fazel-Zarandi et al. 2011, Janowicz, Raubal & Kuhn 2011).
Software Modeling. So Class Diagram and ER Diagram, Database Modelling = 4 (see Figure 1).

Without considering the weights of edges, the distance from Class Diagram to Database Modelling Distance\(_T\)\(_J\) (Class Diagram, Database Modelling) = 1. Obviously the distance is shorter than the distance between UML and Database Modelling in the previous ontology hierarchy \(T_1\), i.e. Distance\(_T\)\(_J\) (UML, Database Modelling) = 4 (see Figure 1).

As we see in Example 4.2, quality of ontology hierarchies used for matching affects the results of competence matching. Our approach of updating ontology hierarchy can lead to better matching results for the motivation example in Section 2, i.e. \(S_1\) could be selected using the revised ontology hierarchy.

5 Related Work and Discussion

A survey conducted in (Sertkaya 2011) groups the works on bridging the gap between FCA and DLs into two categories, with some work on enriching the language of FCA by borrowing constructors from DL languages (Prediger & Stumme 1999) while some other work on employing FCA methods to solve problem encountered in knowledge representation with DLs.

FCA has been used in building subsumption hierarchies. Most works in the literature apply FCA to build subsumption hierarchies from scratch. In (Baader & Sertkaya 2004), FCA is applied to build subsumption hierarchies with given set of description logic. (Prediger & Stumme 1999) extracts data from relational databases to construct a formal context by using conceptual scales. DLs and attribute exploration is used to specify which attributes cannot occur together and DL reasoner is used during the attribute exploration process as an expert to answer the implication questions and provide counterexample whenever the implication does not hold. In (Baader 1995) FCA is used to compute an extended subsumption hierarchy by building a formal context of a give set of DL concepts where attributes of the formal context are the defined DL concepts. Attribute exploration is used to compute the subsumption hierarchy of all least common subsumers of a given set of concepts.

FCA has also been used for merging ontology hierarchies in (Bendaoud, Napoli & Toussaint 2008, Cur & Jeansoulin 2008, Stumme & Maedche 2001). However all of them assume that sample data sets are given for any give ontologies. Merging of ontology hierarchies is then performed by merging the existing data sets. However, very often we can obtain an ontology hier-
archy but without any corresponding data sets. In this case, the approaches of merging ontology using existing data sets cannot be applied. Our approach of maintaining existing ontology hierarchies does not require an matching data sets and therefore can be applied in all the situations of ontology maintenance. Further, for a given ontology hierarchy our approach can generate a representative data sets, which is of small size, if it is not minimal. The following step of updating ontology using the representative data sets can be performed efficiently as only small number of objects need to be merged. Furthermore, the aim of maintaining and revising existing ontology is to improve semantic searching result. Our approach takes input new knowledge from domain experts and uses FCA to insert new relations and new DL concepts in the existing ontology hierarchies. With the maintaining and revising of ontology hierarchies the results of semantic searching and matching can be improved.

6 Conclusion

In this paper we have proposed an approach for maintaining ontology hierarchies using formal concept analysis. We have proposed a process for updating existing ontology hierarchies together with an algorithm for updating ontology hierarchies. This approach is efficient for updating existing ontologies with new subsumption relations or new concepts. We have shown that the approach can indeed improve existing ontologies so that by using it matching results can be improved.

References


Optimal Selection of Operationalizations for Non-Functional Requirements

Amy Affleck1 Aneesh Krishna1 Narasimaha R Achuthan2

1Department of Computing, 2Department of Mathematics and Statistics
Curtin University
Perth, Western Australia
Email: amy.affleck@postgrad.curtin.edu.au, {A.Krishna,N.R.Achuthan}@curtin.edu.au

Abstract
Several long-standing problems in software engineering are concerned with inadequate requirements elicitation, analysis, specification, validation, and management. This deficit is a major cause of project failure and as such several techniques and frameworks have been developed to assist developers in handling requirements. Methods for handling functional requirements have been in existence for many decades, however methods for handling non-functional requirements are a more recent development.

The Non-Functional Requirements (NFR) Framework is one such method that models non-functional requirements and associated implementation methods. This paper extends the previous quantitative reasoning extension into a single objective optimisation model. The model aims to selectively choose operationalizations in order to increase the overall satisfaction of non-functional requirements. Additionally, the optimisation model will be able to handle larger and more complicated graphs than the original framework.

Keywords: Requirements Engineering, Non-Functional Requirements, NFR Framework, Optimisation, Simulation

1 Introduction
Software engineering is the application of sound engineering principles to software development in order to economically obtain reliable and efficient software (McIlroy, 1969). A key process in this application is requirements engineering; the careful assessment, definition and justification of the needs that the system must fulfil, including system features and their purpose (Ross & Schuman, 1977; Zave, 1997). Since the development of software engineering it has been claimed that inadequate, inconsistent, incomplete, and ambiguous requirements are a leading cause of projects running over time, over budget, or even failing completely, comprehensive studies have supported this idea (Bell & Thayer, 1976; Boehm, 1984; Brooks, 1987; The Standish Group, 2000; Mairiza et al., 2010).

Goal-Orientated Requirements Engineering (GORE) is a form of conceptual modelling that identifies requirements as goals to be achieved (van Lamsweerde & Letier, 2000; van Lamsweerde, 2001, 2004). These high level, generalised goals are progressively and iteratively refined in order to identify potential implementation methods and programming constructs that result in the goals achievement (Mylopoulos et al., 1999). GORE modelling is increasing in popularity and there are several well known frameworks including i* Framework (Yu, 1995), Knowledge Acquisition in Automated Space (KAOS) (Dardenne et al., 1991), Non-Functional Requirements (NFR) Framework, (Chung et al., 2000), and tropos (Bresciani et al., 2004) that utilise the concept.

GORE models allow developers to represent the requirements and potential implementations, however the graphs are easily subject to scalability issues (Chung et al., 2000; Heaven & Letier, September 2011; Letier & van Lamsweerde, 2004; Liaskos et al., 2010). These issues can decrease readability and make it difficult to determine the best possible decisions, optimisation is one method of overcoming this problem. This paper presents an optimisation model for the NFR Framework that focuses on maximising requirement achievement and is based on previous work to quantitatively support the framework (Affleck & Krishna, 2012). The model’s soundness and completeness, that is its ability to correctly handle any Softgoal Interdependency Graph is verified through simulation based analysis.
The remainder of this paper is structured as follows; section 2 briefly outlines the NFR Framework and previous quantitative extension. Section 3 then presents an optimisation model which section 4 implements and analyses. Section 5 discusses a range of related work and finally section 6 concludes the main points of this paper and lays out several ideas for future work.

2 The NFR Framework & Background

The NFR Framework (Chung et al., 2000) presents a well documented systematic method of handling decisions in regards to non-functional requirements during late requirements and early design. The non-functional requirements are presented as softgoals in a Softgoal Interdependency Graph that is progressively refined until the leaf-softgoals are clear and free of ambiguity. Leaf-softgoals are the lowermost nodes of the graph, that is they have no children of their own. Methods of achieving these leaf-softgoals are then identified and the relationships defined by a qualitative relationship; these methods are called operationalizations and like softgoals can also be refined. Once the Softgoal Interdependency Graph has been constructed, developers examine the model and use it as a tool to decide which operationalizations should be implemented. These decisions are then propagated to the leaf-softgoals and softgoals to provide a qualitative idea of their effect. Figure 1 outlines an exemplar of the NFR framework taken from Chung et al. (2000, Chapter 2). Due to space restrictions readers are directed to Chung et al. (2000) for further details on the NFR Framework.

Previous work introduced the idea of extending the NFR Framework to support quantitative reasoning; including the quantitative propagation of decisions (Affleck & Krishna, 2012). The proposed extension was based on ideas and methods presented in several papers including Feather & Cornford (2003); Giorgini et al. (2003); van Lamsweerde (2009); Letier & van Lamsweerde (2004); Liaskos et al. (2010). The extension is based on the format of the original framework with several key differences. First the leaf-softgoals are weighted between 0 and 1 according to their priority — a previously qualitative note. Next the relationships between operationalizations and leaf-softgoals (impacts) are given a value between -1 and 1, again a conversion of qualitative data. Once this data has been gathered a score is calculated for each operationalization based on the associated impact values and leaf-softgoal weights. The operationalizations are accepted or rejected based on their scores and decompositions. Once the operationalizations have been accepted or rejected, the previous qualitative propagation algorithm is replaced with a quantitative version based on the values of the impact relationships. Finally the weights of the leaf-softgoal and their scores are used to create an ideal attainment score for the system and an actual attainment score for the implementation. The attainment score is a measure of how well the leaf-softgoals have been achieved. Figure 2 shows the results of applying the extension to the previous exemplar. For the remainder of this paper the extension is referred to as the Extended NFR Framework. Due to space restrictions readers are directed to Affleck & Krishna (2012) for further details.

Previous analysis of test results from the Extended NFR Framework revealed that an optimisation model could improve the results in several ways. This includes increasing the overall attainment score, decreasing negative leaf-softgoal scores, and reducing the number of accepted operationalizations. The next section presents an optimisation model for the NFR Framework that aims to satisfy these goals.

3 The Optimisation Model

This section outlines an optimisation model for the NFR Framework by first expressing the Softgoal Interdependency Graph as a directed graph, then defining associated constants and variables. Once the graph is defined the calculations for the given variables are laid out and finally the graph data is used to construct an objective function.

The relationships between softgoals, leaf-softgoals, and operationalizations are modelled using the directed graph $G = (N, A)$. The definitions of the set of nodes $N$ and the set of arcs $A$ are detailed below.

$N = SG \cup LSG \cup OP$ where $SG$, $LSG$, and $OP$ are respectively the set of softgoals, the set of leaf-softgoals, and the set of operationalizations. $A = A_1 \cup A_2 \cup A_3 \cup A_4$ where $A_1$, $A_2$, $A_3$, $A_4$ are defined as follows.

1. $A_1 = \{(i,j) : i \in SG, j \in SG \text{ and } j \text{ is a decomposition (ie: child) of } i \}$
2. $A_2 = \{(i,j) : i \in SG, j \in LSG \text{ and } j \text{ is a decomposition (ie: child) of } i \}$
3. $A_3 = \{(i,j) : i \in OP, j \in OP \text{ and } j \text{ is a decomposition (ie: child) of } i \}$
4. $A_4 = \{(i,j) : i \in LSG, j \in OP \text{ and the operationalization } j \text{ contributes to leaf-softgoal } i \}$

Input constants defining node details and relationships between nodes for the graph $G$ are expressed using the following conventions:
I For any node $i \in N$ let $C_{(i)}$ be the children of node $i$, for $(i, j) \in A_1 \cup A_2 \cup A_3$ such that $j \in C_{i}$, the relationship is classified by either an AND operation or OR operation.

II For any node $i \in LSG$ there is a weight denoted by $LSG_{w_{(i)}}$, this weight represents the priority of the node and can be in the range 0 to 1. A weight of 1 represents a critical goal whereas a weight of 0 represents a superfluous goal.

III For any node $(i, j) \in A_1$ there is an associated impact value, denoted by $impact_{(i,j)}$ and its value is between -1 to 1. The impact models the effect an operationalization has on a leaf-softgoal.

The variables of graph $G$ are also dependant on several auxiliary variables that are utilised in the equations outlined further down. These auxiliary variables are defined as:

I For any node $i \in LSG$ there is an associated variable $LSGs_{1_{(i)}}$ that can take any real value. This variable represents the raw score before imposing the lower and upper limits.

II For any node $i \in LSG$ there is an associated variable $LSGs_{1_{(i)}}$ that can take any real value. This variable represents the raw score before imposing the lower and upper limits.

III For any node $(i, j) \in A_1$ there is an associated decision variable denoted by $OP_{a_{(i)}}$ that can take the value of 1 (or 0), if the operationalization is accepted (or rejected).

Given a Softgoal Interdependency Graph defined as graph $G$ by the aforementioned constants the variables of the graph such as operationalization acceptance and leaf-softgoal scores can be calculated as follows.

I The leaf-softgoal score ($LSGs_{1_{(i)}}$) is calculated according to equation (1). The calculation is based on the values of $impact_{(i,j)}$ and $OP_{a_{(j)}}$ where $j \in C_{(i)}$.

$$LSGs_{1_{(i)}} = \sum_{j \in C_{(i)} \cap A_4} (impact_{(i,j)} \cdot OP_{a_{(j)}})$$

Figure 1: The final Softgoal Interdependency Graph for the Bank System Example — NFR Framework
The selection of operationalizations \( \text{O} \text{P}_a(i) \) is calculated by equations (2) and (3).

(a) The first equation (2) calculates the selection of operationalizations corresponding to an AND decomposition. Equation (2a) works top down to force all nodes \( j \in C(i) \) for the parent node \( i \), to be accepted if the parent should be accepted. Equation (2b) checks that if all nodes \( j \in C(i) \) have been accepted due to the contributions made by \( \text{impact}(x,j) \) then as a side effect node \( i \) is also accepted.

(b) The second equation (3) calculates the selection of operationalizations corresponding to a OR decomposition. Equation (3a) works top down to force at least one node \( j \in C(i) \) for the parent node \( i \), to be accepted if the parent should be accepted. Equation (3b) checks that if a node \( j \in C(i) \) has been accepted due to the contributions made by \( \text{impact}(x,j) \) then as a side effect node \( i \) is also accepted.

For a node \( i \in O \) with AND operation

\[
|C(i)| \text{O} \text{P}_a(i) \leq \sum_{j \in C(i)} \text{O} \text{P}_a(j) \tag{2a}
\]

\[
\sum_{j \in C(i)} (\text{O} \text{P}_a(j)) - \text{O} \text{P}_a(i) \leq |C(i)| - 1 \tag{2b}
\]

For a node \( i \in O \) with OR operation

\[
\text{O} \text{P}_a(i) \leq \sum_{j \in C(i)} \text{O} \text{P}_a(j) \tag{3a}
\]

\[
\text{O} \text{P}_a(i) \geq \text{O} \text{P}_a(j) \quad \forall (i, j) \in A^3 \tag{3b}
\]

Given the variables and constants of the graph, the ideal attainment score (denoted by \( A_{\text{ideal}} \)) can be calculated as equation (4). The ideal attainment score represents the ideal system where all leaf-softgoals have been satisfied to their full extent. The calculation is based on weight so higher priority leaf-softgoals have a larger effect on the score.

\[
A_{\text{ideal}} = \sum_{i} \text{LSG}w(i) \tag{4}
\]

To compute the actual attainment score (denoted by \( A_{\text{actual}} \)) the \( \text{LSG}w_{1(i)} \) computed in equation (1) must be transformed to the variable \( \text{LSG}w_{1(i)} \) as fol-
The actual attainment score is calculated as equation (6). This score takes into account the leaf-softgoal scores as they are an individual measure of how well the leaf-softgoal has been satisficed.

\[
A_{\text{actual}} = \sum_{LSG_i \leq -1} ((-1) \cdot LSGw_i) + \sum_{-1 < LSG_i < 1} (LSGw_i \cdot LSG_i) + \sum_{LSG_i \geq 1} ((1) \cdot LSGw_i) \quad (6)
\]

Note that whenever \( LSG_i \geq 1 \), the contribution to \( A_{\text{actual}} \) from the third term of equation (6) is the same as the contribution to the ideal attainment score in equation (4). On the other hand when \( LSG_i < 1 \), the contribution from the first two terms in equation (6) is less than the ideal attainment score equation (4). To overcome this issue two variables are introduced \( n(i) \) and \( p(i) \) that when compared to the ideal score of 1 respectively represent the deficit and surplus equation (7) (Ignizio, 1978). These variables are constrained by equations (8) and (9)

\[
\sum_i LSGs_{i(j)} + n(i) - p(i) = 1 \quad (7)
\]

\[
n(i) = 0 \iff LSGs_{i(j)} \geq 1
\]

\[
p(i) = 1 - LSGs_{i(j)} \iff LSGs_{i(j)} < 1 \quad (9)
\]

While maximising \( A_{\text{actual}} \) of equation (6) by choosing a negative penalty as the objective function coefficient for the variable \( n(i) \), the model can ensure that the following conditions are satisfied:

\[
n(i) \geq 0 \quad (8)
\]

\[
p(i) \geq 0 \quad (9)
\]

The closer the value of \( n(i) \) is to zero the larger the contribution of \( LSGw \) to the actual attainment score. Hence minimising the value of \( n(i) \) for each leaf-softgoal results in a higher attainment score; this gives an objective function equation (10) that approximates the optimal actual attainment score.

\[
\min \sum_i (n(i) \cdot LSGw_i) \quad (10)
\]

4 Simulation and Evaluation

In order to evaluate the optimisation model and compare it to the previous Extended NFR Framework, a simulation was created using C# and LPSolve. C# was chosen due to experience and availability, LPSolve was chosen for many reasons. LPSolve is a free Mixed Integer Linear Programming (MILP) solver licensed under the GNU Lesser General Public License, meaning full source code, examples, and manuals are available at no cost. It can solve models comprising of pure linear, mixed integer/binary, semi continuous, and special ordered sets. Additionally LPSolve can be called from a variety of both programming and mathematical languages. LPSolve also has the ability to read and process models created from a range of other solvers.

The simulation takes as input a graph definition consisting of all previously defined constant values, and outputs the value of \( OP_{a(i)} \) for every operationalization and the value of \( LSGs_{i(j)} \) for every leaf-softgoal. The simulation was executed with 5 sets of randomly generated test data designed to maximise the combinations of constants used. Each of the test cases was sketched by hand and the results of the simulation applied to ensure correct handling of operationalization decomposition; due to space restrictions and the size of the test cases only Test Case 5 is included here. Figure 3 shows the results of the Extended NFR Framework on a simplified Softgoal Interdependency Graph; due to readability the values of the impacts are not shown. Figure 4 shows the same test case but with the results of the optimisation model, in both cases the accepted operationalizations have been highlighted. While the score changes appear minimal, the following discussion shows significant improvement in the resultant attainment score.

As previously outlined the goal of creating an optimisation model is threefold; first is to improve the attainment score; second is to decrease the denied softgoals; third is to selectively accept operationalizations to prevent superfluous implementation. The following discussion will assess the results of the five test cases according to these criteria, the results of the original Extended NFR Framework are used as a benchmark for the comparison.

The first goal is to create a more selective approach to operationalization selection as the associated variables \( OP_{a(i)} \) function as decision variables in the optimisation model. Table 1 outlines the number of operationalizations present in each test case and how many were accepted for the original extension and the optimisation. This table shows the optimisation provides a significant reduction to the number of accepted operationalizations across all five test cases. Further discussion will show that the reduction was not to the detriment of leaf-softgoal and attainment scores. Though it cannot be proven that the reduced set of accepted operationalizations would lower cost and effort, there is a correlation between the two.

Figure 5 shows the leaf-softgoal score distributions
Figure 3: Softgoal Interdependency Graph after applying the Extended NFR Framework to Test Case 5

Figure 4: Softgoal Interdependency Graph after applying the optimisation model to Test Case 5
from the original Extended NFR Framework at intervals of 0.1, the key points of this graph are -1 and 1 intervals, and the frequency of positive and negative scores. Dividing the scores in this manner allows the grouping of scores at the -1 and 1 mark to become apparent. In comparison to score distribution from the optimisation model (figure 6), the later shows a significant shifting of scores from negative to positive. This includes a small reduction of the denied scores ($LSGs = -1$), and increase of the satisfied scores ($LSGs = 1$). While the number of scores at the outermost limits has not been significantly altered, the increased magnitude of positive scores, especially those above 0.5, reflect an overall improvement. While these graphs clearly show an improvement across leaf-softgoal scores due to the inclusion of leaf-softgoal weights in attainment calculations, it can not be claimed that this increase would significantly alter the actual attainment scores.

As established previously, operationalizations have been selectively accepted, hence the leaf-softgoal scores were able to improve. The improvement occurs as once a priority softgoal exceeds a score of 1 operationalizations with negative effects on the softgoal can be accepted if the total score `evens out’ at 1. In this manner operationalizations with negative effects on priority softgoals and positive effects on non-priority softgoals can be accepted, resulting in a higher attainment score, given the negative effect has been countered. The Extended NFR Framework does not cater to this situation as priority softgoals tend to dominate the operationalization score and hence its selection.

The main purpose of the optimisation model is to improve the attainment score for the specified Softgoal Interdependency Graph. Table 2 outlines the ideal attainment score compared to the actual attainment score for both the Extended NFR Framework and the optimisation model. These scores show improvements ranging from 7.56% to 41.62%. Similar to the leaf-softgoal scores, a comparison of the attainment score improvements and operationalization selection demonstrates that decreasing the number of selected operationalizations does not negatively impact the system.

The application of the optimisation model is aimed at situations with large Softgoal Interdependency Graphs where the number of relationships makes decisions difficult to attain purely by developer judgement; in this manner the optimisation model attempts to overcome the scalability issues present in the original NFR Framework. The obvious application of the optimisation model is to determine an optimal operationalization set when the Softgoal Interdependency Graph involves contradictions, the model can also be applied to determine the optimal set among complimentary alternatives. However the development of an easy to use tool to apply the optimisation is vital to its successful implementation.

### 5 Related Work

Since the concept of goal modelling emerged, several frameworks have been proposed that take advantage of its principles. This section briefly explains a handful of these frameworks and the quantitative and optimisation extensions that they have inspired. While developing quantitative support for these frameworks and models is a popular line of research, significantly less work has been done to introduce the notion of optimisation.

The i*' Framework (Yu, 1995) is applied during the early phases of system modelling in order to gain an understanding of the problem domain. The models created by the i*' Framework outline various softgoal dependencies, goal dependencies, task dependencies, and resource dependencies between actors; models can also outline the rationale behind individual actors. The idea of the i*' Framework is to model the entire process of the system not just the software components, the framework is used for requirements elicitation and is qualitative in nature.

The basic KAOS model (Dardenne et al., 1991) is a qualitative approach to requirements elicitation that extracts requirements from goal diagrams.

---

**Table 1: Accepted operationalizations for the Extended NFR Framework and optimisation model**

| Test Case | $|OP|_{i}$ | ENFR $|OP_{a(i)} = 1|$ | Optimisation $|OP_{a(i)} = 1|$ |
|-----------|-----------|----------------|-------------------|
| Test Case 1 | 42 | 33 | 6 |
| Test Case 2 | 16 | 9 | 6 |
| Test Case 3 | 37 | 24 | 10 |
| Test Case 4 | 44 | 30 | 17 |
| Test Case 5 | 16 | 14 | 11 |

**Table 2: Attainment scores for the Extended NFR Framework and optimisation model**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>$A_{ideal}$</th>
<th>ENFR $A_{actual}$</th>
<th>Optimisation $A_{actual}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case 1</td>
<td>12.9</td>
<td>3.697</td>
<td>9.066</td>
</tr>
<tr>
<td>Test Case 2</td>
<td>14.7</td>
<td>3.776</td>
<td>5.278</td>
</tr>
<tr>
<td>Test Case 3</td>
<td>11</td>
<td>7.449</td>
<td>8.281</td>
</tr>
<tr>
<td>Test Case 4</td>
<td>14.7</td>
<td>7.542</td>
<td>12.873</td>
</tr>
<tr>
<td>Test Case 5</td>
<td>7.1</td>
<td>3.122</td>
<td>4.096</td>
</tr>
</tbody>
</table>
Figure 5: Leaf-Softgoal score distributions for the Extended NFR Framework across five test cases

Figure 6: Leaf-Softgoal score distributions for the optimisation model across five test cases
These goals can contain a combination of both functional and non-functional system aspects, the models include methods of achieving these goals. KAOS is purely concerned with the goals of the system not actors or actor interactions.

The core idea of Tropos is modelling agents and goals is a method similar to i* however the approaches differ as Tropos presents a requirements-driven method of software development (Bresciani et al., 2004). The concepts used are not limited to requirements engineering instead persist through the entire software life-cycle. As with i* and KAOS, tropos presents a qualitative approach.

While these frameworks present comprehensive approaches to goal modelling and are valid in their own right, a considerable amount of research into developing quantitative support has been completed.

van Lamsweerde (2009) applies the qualitative propagation algorithm utilised by the NFR Framework and first introduced in Chung & Nixon (1995) to the KAOS Framework before developing a lightweight quantitative alternative based on comparative values. The initial approach is improved by the introduction of gauge values which are derived from the requirements specification.

Giorgini et al. (2003) introduces quantitative reasoning with goal models, the proposed method is based heavily on first order logic and mathematical reasoning. While this method requires strong background knowledge it is comprehensive.

Liaskos et al. (2010) introduces an in-depth approach to goal priorities creating optional and mandatory classifications. A quantitative approach is then defined that uses the former category to evaluate alternative designs for the latter, the approach also introduces temporal constraints to increase the accuracy of the goal model. The proposed approach is implemented using a preference based planner to create a set of optimal solutions.

Letier & van Lamsweerde (2004) develops a quantitative extension to the KAOS framework that includes partial goal satisfaction and probabilistic theory. The main goal of this work was to overcome the limitations of conclusions and accuracy in the qualitative framework. The work includes the presentation of a systematic method for elaborating goal models based on the developed techniques. Heaven & Letier (September 2011) extends this work to overcome scalability issues by presenting automated techniques for simulating the previous quantitative models and developing a multi-objective optimisation algorithm to explore the alternative designs.

6 Conclusions and Future Work

This paper proposed an optimisation model for the NFR Framework that attempted to increase the overall system attainment. The model presented a mathematical network representation of a Softgoal Interdependency Graph, this network was then developed to create an objective function. The proposed optimisation model was then implemented and tested across five test cases. Analysis of these tests was based on improvements to the leaf-softgoal scores, attainment scores, and operationalization selection. The analysis effectively demonstrated that the optimisation model notably improved the previous Extended NFR Framework.

The work presented in this paper can be further improved in several ways. The first and possibly most vital step is to develop a formal proof for the objective function to remove the variability of approximation. The next step is to develop a multi-objective approach to the NFR Framework that allows factors such as cost and effort to be simultaneously considered (Ignizio, 1976; Lee, 1972; Chen, 1998).

Additional work would also integrate several quantitative handlers for the NFR Framework into a single tool allowing developers to explore different solutions or confirm a single solution across a range of methods with a minimal increase in required effort.

References


Heaven, W. & Letier, E. (September 2011), Simulating and optimising design decisions in goal models, in ‘Proceedings of 19th IEEE International Requirements Engineering Conference (RE 2011), Trento, Italy’.


A Framework for Cost-Aware Process Management: Generation of Accurate and Timely Management Accounting Cost Reports

M. T. Wynn\textsuperscript{1} W. Z. Low\textsuperscript{1} W. Nauta\textsuperscript{2}

\textsuperscript{1} Business Process Management Discipline, Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia. Email: \{m.wynn@qut.edu.au, w4.low@qut.edu.au\}

\textsuperscript{2} Information Systems Group, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands. Email: \{w.e.nauta@alumnus.tue.nl\}

Abstract

Organisations are constantly seeking efficiency improvements for their business processes in terms of time and cost. Management accounting enables reporting of detailed cost of operations for decision making purpose, although significant effort is required to gather accurate operational data. Business process management is concerned with systematically documenting, managing, automating, and optimising processes. Process mining gives valuable insight into processes through analysis of events recorded by an IT system in the form of an event log with the focus on efficient utilisation of time and resources, although its primary focus is not on cost implications. In this paper, we propose a framework to support management accounting decisions on cost control by automatically incorporating cost data with historical data from event logs for monitoring, predicting and reporting process-related costs. We also illustrate how accurate, relevant and timely management accounting style cost reports can be produced on demand by extending open-source process mining framework ProM.

Keywords: Business process management, management accounting, process mining, cost mining, cost reporting.

1 Introduction

Organisations are constantly seeking efficiency improvements for their business processes in terms of time and cost. Enterprise cost reduction and business process improvement both emerge as important topics in Gartner’s 2011 survey amongst more than 2000 CIO’s, ranking at places 3 and 5 respectively (Gartner 2011).

Business process management (BPM) provides an innovative approach to solving long-standing business challenges. This is achieved by promoting a process-centric view of an organisation through end-to-end management of business processes. An executable process model (or a workflow) contains descriptions of what tasks need to be performed, when and by whom, what information they require and what information they produce (Weske 2007). A BPM system (BPMS) supports planning, execution, (re)design and deployment of workflows (Weske 2007). Process mining provides techniques to discover, monitor and improve processes by extracting knowledge from event logs (van der Aalst 2011). A large number of process mining techniques exist that can discover what really happened during process execution using timing and resource information etc. but there are no process mining techniques that can directly support the cost-perspective as of yet. An increased uptake of BPMSs in recent years has increased the potential for real, evidence-based analysis of the cost associated with the execution of business processes in a timely manner.

1.1 Problem Definition

The motivation for this work comes from challenges encountered in the Management Accounting discipline, some of which could be alleviated by making use of detailed process knowledge available from the BPM discipline.

Management accounting discipline explores ways to capture the real cost of operations accurately and processes this fine-grained cost information into a form suitable for their planning and control decisions making as highlighted in the reports by International Federation of Accountants (Professional Accountants in Business Committee 2009a,b). Based on a detailed literature review and interviews with a number of management accounting experts, the following two key characteristics of the discipline are identified.

Multiple management accounting techniques: Unlike the financial accounting discipline, whereby the accounting techniques used for external reporting are prescribed by legislation, an organisation has flexibility to use any suitable accounting technique for internal decision making. One of the popular management accounting techniques is activity-based-costing (ABC) which emphasises on the per (activity) unit cost of all possible activities (Cooper & Kaplan 1988). However, the ABC technique requires a substantial effort to implement and to be kept up-to-date (Kaplan & Anderson 2003).

A new technique: time-driven ABC (TDABC) was introduced to address the shortcomings of the ABC technique. TDABC focuses on the per (time) unit cost of all possible activities (Kaplan & Anderson 2003). In addition to the TDABC technique, another contender for next generation cost management systems is Resource Consumption Accounting (RCA) (Tse & Gong 2009, White 2009). RCA highlights the per unit cost of the resources that are involved in process execution. The RCA technique emphasises the resource perspective and is interested in determining the over/under utilisation of...
resources (White 2009). In most cases, management accountants tailor one or more variants of these techniques to produce their management accounting reports based on an organisation’s needs.

**Multiple cost reporting templates.** As each management accounting technique looks at different aspects of cost at different levels of an organisation, there is no standard format for management accounting cost reports which is in contrast to the standard format available for external financial reports such as balance sheets and profit and loss reports.

As the main purpose of these management accounting reports is for operational decision making, the reports are of limited use to managers if the data contained in them cannot be relied upon and if the reports are produced “too late, too aggregated or too distorted” (Johnson & Kaplan 1987). Hence, the following are two main challenges concerned with data collection.

**Information gathering is time consuming.** Management accountants who design, use and collect cost information typically work with others from many different parts of an organisation as they need to delve below the information recorded in financial ledgers and external financial reports to produce management accounting reports (Professional Accountants in Business Committee 2009a). Information gathering for decision making remains a costly and time-consuming exercise for organisations.

**The need for accurate cost and process information.** Management accountants typically work with budget figures (e.g., volume of raw materials required and their associated costs, volume of production etc.) which are then compared against the actual production figures/expenditures as part of the variance analyses carried out after the fact. There is a huge reliance on the accuracy of the budget figures as operational decisions are being made based on these figures, rather than the actual figures. It is also recognised that in many cases variance analysis highlights a significant discrepancy between budget and actual figures. Hence, access to accurate data in a timely manner is one of the key challenges facing the discipline.

The premise of this paper is that smart automation of the principles behind management accounting techniques may serve to lessen the cost of data gathering and provide timely data for decision making. Provided that an organisation uses an IT system or a BPM system which records historical information about what is happening during the execution of the organisation’s processes in a detailed way, the detailed operational information that a management accountant needs for cost control is available in some form.

### 1.2 Research Goals

Although the two fields of management accounting and business process management look at different aspects of an organisation, there is a huge potential for managing cost in a structured manner based on an explicit link with business processes, since anything done through the application of BPM has an impact on processes and resources within an organisation which in turn influences cost and therefore management accounting decisions.

In the field of BPM, organisations focus on systematically documenting, managing, automating and optimising their business processes. Process logs typically contain detailed records of activities carried out by resources within an organisation which is then analysed to identify suitable process improvement opportunities.

The main goal of our work is to devise an approach to incorporate relevant cost data into the historical information recorded during process executions to provide accurate, relevant and timely information for decision making.

The innovative nature of this approach is that by utilising detailed process knowledge from IT/BPM systems, the approach makes it possible to generate insightful process-based cost reports in an ad-hoc and timely manner. By providing the ability to generate detailed cost reports automatically at any time (e.g., every second/minute should an organisation wishes to do so), the approach also enables a rare opportunity to monitor cost and resource utilisation within process executions in a cost efficient manner.

The main contributions of the paper are three-fold: an architecture for the cost mining framework, the proposed data definitions for incorporating the cost perspective in BPM systems, and a first implementation of the cost reporting functionality in the process mining framework, ProM.

The remainder of the paper is organised as follows: first, we describe the related work to our approach. Next, we present the proposed cost mining framework. This is followed by a discussion on the implementation of the cost mining framework and a number of cost-related plug-ins within the Process Mining framework, ProM, together with the evaluation strategy used for validation. In conclusion, a brief summary together with a discussion on the future work is presented.

### 2 Related Work

Our work combines aspects of management accounting, business process management, and process mining. Some of the most relevant contributions from these broad areas are reviewed below.

Management accounting techniques provide managers with an ability to plan ahead of time and to make informed strategic and operational decisions based on cost (Johnson & Kaplan 1987). The ABC costing technique provides an accurate way of assigning the costs of indirect and support resources to activities, business processes, products, services and customers (Kaplan & Atkinson 1998). Fixed, variable, and overhead costs are taken into consideration and different cost types such as resource costs and activity costs identify appropriate cost drivers. Total cost is then assigned to activities or product categories, which is then further assigned to individual products or services (Cooper & Kaplan 1988). Due to the time-consuming nature of data required for the ABC technique, another technique (time-driven ABC) that decreases the amount of data needed was proposed. The TDABC technique identifies fixed, variable, and overhead costs as well, but instead of assigning cost to individual products or services, it was assigned to a time frame and cost per time unit is calculated (Kaplan & Anderson 2003). Hence, TD-ABC only requires estimates of capacity of committed resources, their cost rates, and activity duration. Another contender for next generation cost management systems is Resource Consumption Accounting (RCA) (Tse & Gong 2009, White 2009). The RCA technique emphasises the resource perspective and is interested in determining the over/under utilisation of resources (White 2009). Based on the cost rate of a resource, idle costs are attributed to those resources that are underutilised. Furthermore, time or capacity-related information (for example, activity duration, utilised resources, and maximum allo-
cated resource capacity), is essential for utilisation-based cost models such as the idle resource cost report. The RCA technique can be viewed as evolution of the ABC technique in Enterprise Resource Planning (ERP) systems (Keys & Merwe 2001). However, information gathering for the purpose of cost reporting and control remains time and resource intensive.

Business process management, on the other hand, defines a methodology and tools to design, configure, execute and diagnose structured and less structured processes within organisations (van der Aalst et al. 2003, Weske 2007). Through the iterative application of BPM techniques, processes are improved in terms of quality, flexibility, time and/or cost (Reijers & Mansar 2005). Process mining facilitates continuous process improvement by extracting knowledge from event logs (van der Aalst 2009, IEEE Task Force on Process Mining 2011). An event log is a data store where a historical record of process execution is kept. Information such as task events (offer, start, complete), data attributes, utilised resources and task durations can be extracted from an event log (van der Aalst 2011). Depending on the level of details recorded in event logs, various analyses of the four different perspectives (control, organisation, case, and time) are possible. As a result, process mining techniques can provide valuable insight into control flow dependencies, data usage, resource utilisation and various performance-related statistics.

Process mining is enabled through the facilitation of software. One of the leading tools for process mining is the ProM framework (van der Aalst et al. 2007). ProM provides a generic open-source framework for process mining/analysis tools in a standard environment and plug-ins can be added to extend its functionalities (Verbeek et al. 2010).

There is also a body of work that brings the notions of cost and processes together. Cost has always been one of the key factors under consideration in terms of process improvements (Reijers & Mansar 2005), and process simulation (Laguna & Marklund 2005), and within the manufacturing domain (Grabski & Marsh 1994, Liebers 1998, Leung 1996, Brinke 2002, Brinke et al. 2004). Irani et al. discussed eight different cost taxonomies for IT/IS-related cost (Irani et al. 2006). Mutschler et al. presented a qualitative cost analysis methodology to better understand the cost of Process-Aware Information Systems development projects (Mutschler et al. 2007). There is also a body of work that focuses on measurement and aggregation of Quality of Services in workflow and web services including cost (Jaeger et al. 2004, Cardoso et al. 2004, Mohabbati et al. 2011).

Since the introduction of ERP systems, studies have been conducted on the effects of ERP systems on traditional management accounting practices (Booth et al. 2000, Granlund 2001, Granlund & Mallin 2002, Grabski et al. 2009, Hyyönen 2010). Vom Brocke et al. proposed an information model to link the ARIS accounting structure with ARIS process semantics using Event Driven Process Chains (EPC) (vom Brocke et al. 2011). This illustrates a growing interest in this interdisciplinary area of accounting and business process management and motivates us to undertake our research in the area of cost-aware BPM.

3 Cost Mining Framework

3.1 Research Approach

To achieve our objective of making process mining ‘cost-aware’, we followed the research approach involving literature review, architecture design, implementation and validation. Based on the findings from a detailed literature review, we identified a number of key requirements from the management accounting perspective and the business process management perspective. These requirements are then used to develop an integrated approach for process-based cost mining as well as conceptual data models. The approach is then validated by development of a prototype as well as interviews with experts from the management accounting and the BPM disciplines.

Figure 1 illustrates the proposed integrated approach for process-based cost mining. The approach is kept simple and high-level to enable both disciplines to see the potential of an integrated approach. The approach also highlights the importance of information sharing between management accountants and BPM professionals.

The top half of the diagram depicts the important role played by management accountants in providing accounting knowledge to the cost mining framework. This may involve supply of static cost data collated from HRM, CRM or other systems, management accounting techniques used in the organisation as well as cost report templates. The bottom half of the diagram shows how existing knowledge from BPM systems can be leveraged for cost analysis. To perform cost analysis, three separate steps are proposed to be carried out:

- creation of a cost model,
- annotation of event logs with cost information, and
- analysis of process-based costs.

This is achieved by making use of process knowledge from BPM systems in the form of executable process models, organisational models and event logs. Detailed process-related cost and resource utilisation cost can be automatically calculated and stored within event logs in the form of cost-annotated event logs. Relevant cost reports can be generated in a timely manner using report templates supplied by management accountants. In addition, graphical reports for BPM professionals and other stakeholders can also be generated from the data recorded in cost-annotated event logs. We also envision that the framework will support the ability to predict process-related costs for currently running cases based on these cost-annotated event logs.

3.2 Motivating Example

Here, we describe a simple telephone repair process that is used throughout the paper to explain our approach (see Figure 2). The process starts by registering a faulty telephone sent in by a customer. After registration, the telephone is sent to the Test Department. There, the problem is analysed and its defect is categorised by an employee with the tester role. Once the problem is identified, the telephone is sent to the Solve Department and a letter is sent to the customer to inform him/her about the problem. The Solve Department has two teams. One of the teams fixes simple defects (role: Solver Simple) and the other team repairs complex defects (role: Solver Complex). Once the repair has been completed, the telephone is sent back to the Test Department. After the repaired telephone passes the test, the telephone is sent to the customer...
Figure 1: Overview of the integrated approach to cost mining and reporting

and the case is archived. In addition, it is the policy of the company that it only tries to fix a defect in a particular telephone for a maximum of five times. Furthermore, after the problem with the telephone has been analysed and in parallel with repairing and testing, the customer is informed about the result of the defect analysis.

Figure 2: A BPMN model of the telephone repair process

There are altogether 12 employees in the telephone repair company of whom six work as testers in the Test Department and the other six works as problem solvers in the Solve Department. The organisational model is shown in Figure 3. The resource assignments in the process are role-based and shown with cost data in Figure 4.

Figure 3: The organisational model of the telephone repair company

To be able to calculate cost incurred during activity executions of this process, we need to know relevant cost rates for this process. For instance, we can find out from the HR system the per hour rate for labour costs of an individual employee or a role (i.e., tester, solver). In Figure 4, we assign one or more cost rates for every activity in the telephone repair process. We make use of the notions behind Time-driven ABC costing, whereby, the cost of an activity is calculated based on the time it takes to carry out an activity by one or more resources. We intentionally use many different types of cost rates for illustrative purpose. It is possible for an activity to have different cost rates for different cost types (e.g., the Analyse defect task has three different cost rates for three different cost types - fixed cost, labour and overheads). We also included simple/static cost rates (e.g., $65 per hour for an employee with the role SolverS) as well as more complex/dynamic cost rates where the cost rate is dependent on the characteristics of a case (e.g., the type of phone and/or the type of defect).

Figure 4: Cost data associated with the telephone repair process
Next, we discuss how this cost data is mapped into a cost model of a process.

3.3 Cost Model

Creation of a cost model requires cost information from management accounting and process and resource information from a BPMS. A cost model is considered as the complete specification of cost-related data for a business process.

The data requirements for cost models are collated from the literature and from conversations with domain experts. Three main costing techniques (ABC, TDABC and RCA) are analysed in terms of informational input and output of each technique. We then generalise these requirements to design the conceptual data model of a cost model artefact (Figure 5). This enables us to design a conceptual data model that can cater for multiple management accounting techniques - the first characteristic mentioned within the problem definition section.

We propose a cost model with three core elements: cost driver, cost function and mapping.

- A **cost driver** defines how cost is associated with one or more process elements (resource, activity, case data) and the cost rate. The rationale behind our conceptual design is to keep the model as general as possible so that we are able to capture both simple and complex cost rates (drivers). The following are some example cost drivers supported in our approach.

  - the cost rate of resource ‘R1’ is $50 AUD per hour (variable/labour cost)
  - each invocation of activity ‘A’ costs $20 AUD per instance (processing cost/ fixed cost)
  - each new process instance has an overhead cost of $100 AUD (overhead cost)
  - the cost rate of a resource with a ‘manager’ role performing activity ‘A’ is $30 AUD per hour (labour cost dependent on a task)
  - the fixed cost of activity ‘A’ with the data attribute ‘D1’ being ‘standard’ is $25 AUD while activity ‘A’ with the data attribute ‘D2’ being ‘premium’ is $50 AUD. (fixed cost dependent on a data attribute)
  - the variable cost rate of two resources ‘R1’ and ‘R2’ working together on an activity ‘A’ with data attribute ‘D1’ being ‘premium’ is $200 AUD per hour. (variable/labour cost of multiple resources working together dependent of a task and a data attribute)

- A **cost function** defines a formula for aggregating various cost elements. It is possible to specify cost functions that incorporate both fixed and variable cost components. Variable cost is defined in the cost driver using a cost function that takes into account the duration of a task or the duration of a resource working on that task and/or relevant case data. For instance, we can specify a cost function as follows: each invocation of activity ‘A’ incurs $20 AUD of cost type fixed cost plus $50 per hour for each resource working on the task (labour cost) plus $10 per kilogram of the raw materials used (materials cost).

- A **mapping** provides a way to relate terms used in management accounting to terms used in a BPMS. The terms used to identify resources, activities, and data may differ within a BPMS and an accounting system (e.g., an employee might have two different identifiers (an employeeID in the HR system and a different username in the IT systems). Mappings are required for every workflow element (task, resource or case data) related to a cost driver.

Figure 5 shows the conceptual cost model, modelled using the Object-Role Modelling version 2 notation. For those readers who are not familiar with the ORM notation, we now briefly discuss the ORM 2 notation used in the paper. An ORM model captures relationships between entities (Halpin 2005). An entity type in the model is depicted as a round-corner rectangle (e.g., Cost Model, Cost Driver, Cost Function). An entity type has an identifier label type (e.g., cost driver ID). An entity type can be associated with one or more label types which are depicted as an oval with dash lines (e.g., CD Name, CD description). An entity type can have relationships with one or more entity types (e.g., “linked to” is a binary fact type between a cost driver and a cost technique and it is modelled using a fact type with two roles connected to the two entities involved). A bar above a fact type represents a uniqueness constraint that applies to that fact type (e.g., one to one, one to many, many to many for binary fact types). A black dot attached to the connector between an entity and a role indicates whether this role is mandatory for that entity (e.g., every cost driver must have a cost rate associated with it). A solid line with an arrow head describes a generalisation/specialisation relationship between two entities (e.g., a resource cost driver is a specialisation of a cost driver entity type).

An XML schema has been defined based on this conceptual model and is used as input in the cost mining framework. Our vision is that a process editor will provide functionality to enter cost data and associate a cost model with a process. At this point in time, an XML document that represents a cost model for a process is manually created.

3.4 Cost Annotation

A cost-annotated event log is one where events recorded in a log (e.g., the start event of a case, start and complete events of a task instance) are enriched with detailed cost information related to that event. The cost annotation itself is an association between a case/task instance, cost type, cost driver and a monetary value together with its currency (e.g., carrying out the approval task for caseID 001 incurs $35 AUD of variable cost type using costdriverID c12). A cost element can be associated at the case level (e.g., to represent an overhead fixed cost of a case) and at the task event level to represent fixed or variable cost incurred during task execution. Variable costs are calculated by multiplying a cost rate with relevant durations derived from timestamps of events in a log.

We support six different duration types: waiting, working, and suspended for tasks and assigned, allocated and busy for resources to cater for advanced resource and task lifecycles supported by BPM systems. The ORM 2 representation for cost annotation is shown in Figure 6.

The concept for a cost annotated log is also realised as an XML schema. XES is an XML-based generic format for capturing event information in a structured way (Günther 2009). The XES format uses the perspective of a case (or trace in XES) for grouping events, whereas the time perspective, control flow perspective and organisational perspective can be implemented as optional extensions of events.
in a case. We proposed a cost extension as another optional extension of events in an XES log. The proposed cost extension was sent out to members of the IEEE taskforce on Process Mining \(^1\) and suggested changes have been incorporated into the format. Figure 7 illustrates the cost-annotated event log in the XES format.

### 3.5 Cost Reporting

Fine-grained cost information from cost annotated event logs can be used for analysis purpose. Cost reports can provide different views (i.e., process, resource, organisational/departmental) of cost data in different formats. A number of key functional requirements have been identified in order to support on-demand cost reporting. They include:

- **Support for well-known management accounting techniques.** As different management accounting techniques focus on different cost aspects, the generated reports must adhere to the management accounting’s reporting requirements. The framework was developed to support the principles of common and well-known management accounting techniques such as ABC, TDABC and RCA and their reporting requirements.

- **Support for customisable report templates.** One crucial difference between management accounting reports and external financial reports is that there is no standard format for a management accounting report. Instead, the report format must suit its purpose within an organisation. To cater for the customisation requirement, we propose the use of report templates that can be tailored for different management accounting reports. The framework uses the XSLT stylesheet technology\(^2\) for this purpose and existing cost report templates can easily be converted to XSLT stylesheets for automatic report generation.

- **Ability to filter report output.** Another requirement is the need to support various filtering and display options. Some parties may desire to filter out unnecessary information, or to assign information into different groupings. Thus, the

\(^1\)http://www.win.tue.nl/ieeetfpm/

\(^2\)http://www.w3.org/TR/xslt
Figure 7: A cost-annotated event log where different cost types were recorded against one of the completed events of the Analyze Defect task.

architecture should support filtering and grouping of the cost report information based on organisational hierarchy (e.g., according to departments or roles), on time (monthly, quarterly, or financial year) and different display options (e.g., tabular and graphical display). This is also supported by XSLT stylesheet where XML data can be formatted, filtered and summarised according to the specification provided in an XSLT stylesheet template.

• Ability to accept multiple file inputs with their designated file formats. In order to generate comprehensive cost reports, extra information such as resource and cost rates are required in addition to cost annotated event logs. Therefore, the ability to input multiple files with different formats is supported in the framework.

• Support for resource utilisation reports. In addition to these general requirements for cost reporting, there is also an additional requirement from the RCA technique regarding the assignment of idle capacity and idle cost to resources (White 2009).

RCA-style cost reports highlight underutilised resources and attribute idle cost to such resources based on operational data. A resource’s idle capacity is calculated by deducting the used capacity from the maximum capacity. In order to calculate the idle resource cost for resources involved in process executions, we need to know the maximum/allocated capacity of a resource. However, it was discovered during the requirements analysis of existing workflow organisational models that such capacity information is not being recorded. Hence, we propose a data model to describe the capacity of resources as shown in Figure 8. Each resource, human or non-human, has a maximum working capacity. The capacity can be consumed within a specific duration or at any desired time (i.e., indefinite duration). For example, James (aresource) can work a maximum of 50 hours (max. capacity) per week (time unit). There are some cases where a resource has an unlimited capacity (i.e., it is not a consumable resource). For example, a workdesk or an operation theatre can have an unlimited capacity. An extension to the organisational model XML schema to incorporate the concept of maximum capacity for resources was proposed.

4 Realisation in the Process Mining Framework

Process mining framework ProM is a well-known open source platform which can be extended with additional plug-ins for process analysis. ProM 6 receives the input of event logs in the Extensible Event Stream (XES) format in order to perform various analyses ( Günther 2009). ProM also allows process mining operations to be performed through chaining (or stacking) of plug-ins. Figure 9 illustrates the architecture to support cost mining and cost reporting in ProM. A first implementation of this framework has been developed in ProM 6. These plug-ins can be found in the “CostBasedAnalysis” package in the ProM code repository.

We used a simulated dataset with 1000 completed cases of the telephone repair process. The state of each case in the repair process is represented by the tasks in the process model as well as four data variables with values which are linked to each case. The data values represent phone type, defect type, the frequency of repairs, and the repair outcome. The log thus contains many records of timestamps for when tasks are started and completed, by whom and the values of data attributes where applicable. A cost model, an organisational model and a number of XSLT templates are defined for this example. A cost-annotated log for 1000 cases is generated after running the cost annotation plug-in with the data set and the cost model. The cost-annotated log is then visualised using different cost reporting plug-ins.

4.1 Cost Annotation Plug-in

The cost annotation plug-in produces an event log produced by a BPMS with cost information supplied from a cost model. The cost model is defined prior to process execution and contains all costs associated with the process in the form of cost drivers. After a task instance completes, all information about the task instance is present in the event log. Cost annotation of task instances is based on the task itself, resource(s) participating in the task instance execution and/or case data variables supplied during execution of the task instance. If based on these criteria, a cost driver is applicable for annotating a completed task instance; the task instance is annotated with cost information. The resulting cost annotated event log in XES format contains all cost incurred during process execution, at the most detailed level of task instances.

http://www.processmining.org/prom/tutorials
4.2 Cost Reporting Plug-ins

A number of cost reporting plug-ins that show the potential for using cost annotated logs for reporting purpose have been implemented. Visualisation of cost associated with process executions in the form of charts and graphs showing cost from different perspectives gives a quick overview of cost incurred during process execution. The Cost Visualization plug-in makes use of cost-annotated event logs to visualise resource usage in the form of 2D and 3D pie/bar charts, waterfall charts, and multiple bar charts. Figure 10 illustrates a 2D pie chart generated from the cost annotated event log from the telephone repair example.

Figure 10: A 2D pie chart visualisation of resource utilisation cost for the cost-annotated log of the telephone repair test case example

A number of tabular cost reports have been generated using three variants of the tabular cost reporting plug-in to demonstrate how customised reports can be generated using XSLT templates. A cost annotated log and an XSLT template are mandatory input for all three plug-ins. In addition, a basic filtering/validation function for XSLT templates has been incorporated within these plug-ins.

- The Basic Cost Report plug-in uses as input a cost annotated log and an XSLT template.
- The Enriched (OrgModel) Cost Report plug-in requires an extra input of an organisational model to enable the resulting report to be enriched with organisational information (e.g., departmental/organisational area cost reports can only be generated with additional information found in an organisational model).
- The Enriched (OrgModel & CostModel) Cost Report plug-in supports generation of a report using additional inputs of an organisational model and a cost model. Inclusion of a cost model allows advanced filtering and summarisation to be carried out based on the information in a cost model (such as roles, resource cost driver groupings and cost functions).

Figure 11 shows an example of idle capacity cost report generated from the Enriched (OrgModel & CostModel) Cost Report plug-in. From the report, it can be seen that solvers C1 and C2 are underutilised according to the cost data from the log. The report makes use of an XSLT template that has been designed to generate an RCA style idle capacity cost report. It demonstrates how different data elements from a cost-annotated log, a cost model and an organisational model are collated as part of a cost report.

Figure 11: A screenshot of an idle resource capacity cost report generated from the telephone repair test case example

All these different cost reporting plug-ins illustrate how sophisticated cost analysis/reporting could be conducted by making use of process information obtained from a BPM system together with cost data from an organisation. The proposed cost mining framework enables this by associating cost information with process information through the use of cost.
drivers at design-time and by associating the log information recorded in a BPM system with the corresponding cost model at runtime/post-execution time.

4.3 Evaluation Approach

In terms of evaluation, we are first and foremost interested in demonstrating that the general cost mining framework is valid and that the proposed data models are sufficient and necessary. To this end, we realised our approach as plug-ins of the well-known state-of-the-art Process Mining platform, ProM, and evaluated the implemented framework using an event log of the process that was publicly available together with a complex cost model that we generated for the process. This demonstrates that our approach works as expected and that typical event logs from BPMS systems are suitable for cost analysis. Furthermore, we used time-drive ABC accounting approach for annotating cost information in event logs. A number of resource-based cost reporting plug-ins also illustrated how RCA-style cost reports can be generated.

Next, the proposed XES extension for cost annotations is distributed to members of IEEE Taskforce on Process Mining to get their feedback and consensues on how information can be incorporated into event logs. Their comments were taken into account and changes were made to the XES extension. This further provides assurance that our approach of annotating event logs with cost information is general and that can be used by others to associate cost information with process logs.

To evaluate the suitability and the usefulness of our approach, we interviewed knowledge of both management accounting discipline and the BPM discipline to get their views on the proposed framework, and the data models. All four interviewees provided positive feedback on the research and mentioned that they can see value in the integrated approach to process-based cost analysis. Regarding the framework, interviewees found the framework to be “comprehensive”, “concise”, “systematic”, and “high-level”. In terms of the data models, they mentioned that the concepts captured in the data models are “sufficient” and “more generic that what might be needed”. For instance, interviewees found that cost drivers supported by the framework to be more extensive than typical utilisation rates used in the management accounting discipline (e.g., hourly rates for employees, materials/overhead costs depending on usage), the cost types used in the example (for instance, fixed, variable, overhead) are found to cover the majority of cost types supported in cost reports. In addition, the generic nature of the cost model is also shown to support tailored accounting techniques and tailored cost reports required by the management accounting discipline. In terms of further improvements, one interviewee mentioned the need to demonstrate cost reconciliation between the cost information provided in the reports and the typical cost information available from the general ledger. Another interviewee mentioned that the cost reports could be more generalised, for example, comparison between cost and revenue, budget vs. actual costs, cost comparisons between different processes etc. Overall, this positive feedback provides us with an early indication of the suitability of our approach, although this is subjective in nature.

To further validate the usefulness of our approach, we are looking into carrying out a case study within an organisation that has detailed event logs and makes use of sophisticated cost models for their management accounting decision making.

5 Conclusion and Future Work

There is a huge potential for managing cost in a structured manner based on an explicit link with business processes. In this paper, we focus our attention on developing an overall framework for incorporating the cost perspective in the BPM Systems with the view to enable cost-aware process mining. To this end, we proposed a generic data format for a cost model, a data format for the cost extension in event log (an XES extension for the cost perspective) and illustrated how different management accounting cost reports can be generated using cost annotated event logs. We presented the findings from the first round of evaluations and we plan to carry out a number of case studies to evaluate the proposed framework within an organisational context in the near future.

We acknowledge that potential limitations of the approach can come from the nature of the approach which is bottom-up whereby cost analysis is carried out based on the information recorded in an event log of a process. As a result, there may be a gap between high-level management accounting reports which rely on not-only process-based cost information but other types of information for decision making. We strongly believe that many organisations can benefit from having real-time process-based cost information at their fingertips at design, implementation, monitoring, and evaluation phases of a business process lifecycle.

Many other forms of cost-aware process mining that make use of our proposed cost-annotated logs can be conceived of. For instance, prediction of cost during process execution based on historic data of previous executions is very desirable. Prediction enables anyone involved in the process to make decisions while the process is being executed rather than in the diagnosis phase. Simulation and visualisation of cost information will also enable managers to evaluate their process-related costs in a timely manner. Similarly, incorporating the cost dimension as part of a BPMS solution will enable cost-informed resource allocation and control flow decisions to be incorporated during process execution. We are currently exploring a number of these new research areas, using the work presented in this paper as its foundation.

Acknowledgments. The authors would like to especially thank Wil van der Aalst, Arthur ter Hofstede, Boulewijn van Dongen, Eric Verbeek and Michael Adams for their input to this research. This work is partly supported by an ARC Discovery grant number: DP120101624. The first author is also a research fellow at NICTA Queensland Laboratory.

References


Gartner, C. e. l. r. g. (2011), ‘Reimagining IT: The 2011 CIO Agenda’.


Laguna, M. & Marklund, J. (2005), Business process modeling, simulation and design, Dorling Kindersley (India) Pvt. Ltd.


Liebers, A. (1998), An architecture for cost control in manufacturing; the use of cost information in order-related decisions, PhD thesis, University of Twente.


Professional Accountants in Business Committee (2009a), ‘Evaluating and improving costing in organizations’.

Professional Accountants in Business Committee (2009b), ‘Evaluating the costing journey: A costing levels continuum maturity model’.


Automatic Data Transformation—Breaching the Walled Gardens of Social Network Platforms

Martin Wischenbart\(^2\) Stefan Mitsch\(^1\) Elisabeth Kapsammer\(^1\)
Angelika Kusel\(^1\) Stephan Lechner\(^3\) Birgit Pröll\(^1\) Werner Retschitzegger\(^1\)
Johannes Schönböck\(^2\) Wieland Schwinger\(^1\) Manuel Wimmer\(^2\)

\(^1\) Department of Cooperative Information Systems
Johannes Kepler University, Linz, Austria, Email: \{firstname.lastname\}@jku.at

\(^2\) Business Informatics Group, Institute of Software Technology and Interactive Systems
Vienna University of Technology, Vienna, Austria, Email: \{lastname\}@big.tuwien.ac.at

\(^3\) Netural GmbH, Linz, Austria, Email: s.lechner@netural.com

Abstract

Although many social networks on the Web allow access via dedicated APIs, the extraction of instance data for further use by applications is often a tedious task. As a result, instance data transformation to Linked Data in the form of OWL, as well as the integration with other data sources, are aggravated. To alleviate these problems, this paper proposes a model-driven approach to overcome data model heterogeneity by automatically transforming schemas and instance data from JSON to OWL/XML, utilizing the semantic features of OWL and Jena inference rules. We present a prototypical implementation on the basis of the Eclipse Modeling Framework. This implementation is applied and evaluated on data from Facebook, Google+, and LinkedIn. Finally, we provide prospects for semantic integration and managing evolution, as well as a discussion of how to generalize our approach to other domains and transformations between arbitrary technical spaces.

Keywords: Schema and instance transformation, data model heterogeneity, model driven approach, social network data integration, JSON to RDF & OWL

1 Introduction

In recent years, online social networks have gained great popularity amongst internet users. These networks serve different purposes and communities, for instance, socializing on Facebook or Google+, or establishing professional networks in LinkedIn\(^2\) (Kim et al. 2010). Since users are members of several social networks, integrated profiles from multiple networks are desired to achieve a comprehensive view on users, which would, for instance, increase the quality of personalized recommendations (Abel et al. 2011), or support users’ search activities (Bözson et al. 2012). In our research project TheHiddenU\(^2\) we try to build such comprehensive user profiles in OWL, enriching them with machine learning and information extraction methods, for use in recommender applications. On our platform, non-experts shall express transformations of social network data to their preferred target format. Furthermore, prime goals are to make the system transparent and trustworthy, by (i) providing provenance information whenever demanded, and (ii) by respecting and supporting users’ privacy needs at all times.

For building comprehensive user profiles, in principle, existing user models may be reused as target schemas for the user profiles to be integrated, such as Grapple (Aroyo & Houben 2010) or others (Viviani et al. 2010). However, in contrast to social networks, these approaches often base on common ontologies expressed in OWL (FOAF, etc.\(^3\)). Social networks would benefit from using such ontologies, Semantic Web technologies, and Linked Data, for instance, to solve portability issues and to enable data reuse (Razmerita et al. 2009). Thus, to ultimately extract Linked Data and breach the walled gardens of social networks, the resulting difference in technical spaces demands, as depicted in Fig. 1, that we first tackle technical, syntactic (cf. \(\square\)), and data model heterogeneity (cf. \(\square\)), before structural and semantic heterogeneity (cf. \(\square\)) can be resolved, to finally build integrated user profiles that are (i) complete, concise, and consistent (Bleichner & Naumann 2009), (ii) within aligned ontologies (Parundekar et al. 2010), and (iii) in the form of Linked Data (Heath & Bizer 2011). Dividing these required transformation steps helps to cope with evolution, and facilitates reuse across data sources, because changes are kept local (e.g., JSON replaces XML, while semantic mappings remain unchanged).

For handling all these kinds of heterogeneities, existing tools in the research area of the Semantic Web, such as Virtuoso\(^4\), Triplr\(^5\), and Aperture\(^6\), can be extended with components for user profile extraction. Typically, these components must be configured (i) on the schema level with respect to the target schema, which often is an existing ontology (e.g., FOAF) complemented with a manually created one, and (ii) on the instance level with respect to transformation spec-

---


\(^3\)FOAF: foaf-project.org, sioc: sioc-project.org

Relationship Ontology: vocab.org/relationship

Virtuoso: openlinksw.com

Triplr.org

Aperture.sourceforge.net
Structure of the paper. In the next section, we discuss related research, before in Section 3 and Section 4 we present our approach and an implementation thereof using the Eclipse Modeling Framework. In Section 5 we discuss results and lessons learned from the application on actual user profiles from Facebook, Google+, and LinkedIn. Finally, in Section 6 we give an outlook on integration of user profile data, present prospects for handling evolution of sources and the propagation of changes, and discuss the generalization of the approach for arbitrary source and target technical spaces beside JSON and OWL.

2 Related Work

In this section, we discuss closely related work with respect to instance transformation between JSON and OWL for overcoming data model heterogeneity, ontology engineering, as well as related approaches from the data and model engineering domains.

Concerning the specific transformation of JSON data into RDF graphs or OWL ontologies, most closely related work can be found in terms of so-called RDFizers. Such RDFizers are available for a plethora of different sources, ranging from specific websites over social networks to relational databases and plain files. RDFizers are implemented, for instance, as part of Virtuoso (an enterprise data integration server), Aperture for extracting and querying several information systems in the Semantic Desktop project (Dengel 2007), in d2r and r2rml for publishing relational databases to the Semantic Web (Bizer & Cyganiak 2006, Das et al. 2012), for transforming various microformats into RDF using, for instance, Triplr or Any23, as well as for extracting information from the Old REST API of Facebook (Rowe & Ciravegna 2008) and from Twitter (Mendes et al. 2010). All these frameworks either must be configured manually with respect to data extraction, transformation to RDF, and integration into common ontologies, or simply utilize hard-coded rules for this purpose (specified manually). Also, these approaches often mix the resolving of data model, structural, and semantic heterogeneities, incurring the disadvantages already discussed above. For example, in Virtuoso, so-called cartridges are responsible for extracting data from various sources, and for transforming them into RDF graphs. These graphs utilize the vocabularies pro-

1. www.enterprisearchitect.at
2. www.altova.com/mapforce
3. developers.any23.org
vided by various target ontologies, for instance, FOAF and a complementing Facebook ontology provided by OpenLink, which captures the concepts of Facebook not present in FOAF. For this, Virtuoso requires the target ontologies and XSLT definitions of transformations to these target ontologies, which are often specified manually. Semon (Nuzzolese et al. 2010) goes beyond such tripifiers by focusing on a customizable triplication process. Keeping in mind that data source schemas may frequently change, creating configuration rules-coded rules, and target ontologies manually is a laborious and error-prone task. Our approach, in contrast, separates overcoming different kinds of heterogeneities into sequential transformation and integration steps, also aiming at automating these steps.

Addressing the challenges of ontology engineering, the application of model-driven architectures for development of Semantic Web ontologies has been proposed by (Gašević et al. 2009). An in-depth discussion of meta-models in combination with ontologies for software engineering was done by (Henderson-Sellers 2011), including extensive related work. The method by (Cranefield & Pan 2007) employs Jena rules to create RDF from Mof-based models, whereas (Glimm et al. 2010) use XSLT to generate OWL and UML. The approach of (Glimm et al. 2010) uses OWL2 for meta-modeling, specifying class constraints and relationships as OWL axioms, and synchronizing them with individuals’ role assertions. Our approach combines and generalizes these ontology engineering concepts into a model-driven approach that may be configured to arbitrary source and target technical spaces.

Concerning the individual steps in the transformation process, in particular with respect to schema and instance transformation, in the data engineering domain a considerable amount of research has been conducted. Existing generic approaches map different schemas of the same technical space and exchange data between these schemas (e.g., cf. (Doan & Halevy 2005, Fagin et al. 2009, Legler & Naumann 2007) for surveys on such approaches). More specific approaches (i) map between relational and XML schemas and instances, cf. (Yahia et al. 2004), (ii) map structured sources into RDF, such as (Knoblock et al. 2012, Speiser & Harth 2010), (iii) transform XML to JSON or XML to OWL/RDF (Bohring & Auer 2005, Cardoso & Bussler 2011, Kubeissky et al. 2007, Bischof et al. 2011), and (iv) align the individuals of different ontologies, such as (Noy 2004). To date, however, most approaches rely on manually created and specifically tailored transformation specifications and as a consequence, are vulnerable to evolution of source and target schemas. The requirements mentioned above (schema and metamodelling evolution, platform-independent transformations, etc.) that drove our choice of a model-driven architecture are of minor importance in these references. Being applicable also in presence of schema evolution, especially interesting are the approach of (Bohring & Auer 2005) and CLIO (Fagin et al. 2009, Haas et al. 2005). These approaches generate transformation code in the form of XSLT, XQuery, and SQL queries (depending on the source and target technical space) in order to overcome technical and syntactic, data model, structural, and semantic heterogeneity in a single step. The ideas of such transformation code generation are the basis for the instance transformation step in our model-driven transformation approach for social network data integration, which, as already noted above, separates overcoming different kinds of heterogeneities into sequential steps.

Concerning schema and instance transformation in the model engineering domain, the general idea of bridging several modeling layers within one approach has been proposed, for instance, in the well-known work by (Atzeni et al. 2005). For bridging the meta-model layer, a so-called supermodel has been proposed allowing to transform schemas between different technical spaces, such as OWL, XML, and XSD. For actually transforming the corresponding instances thereof, so-called down functions have been realized (Atzeni et al. 2006), allowing to transform meta-model definitions into target ontologies.

In contrast, we exploit the inference capabilities of OWL, and thus, need not create instance transformations from schema transformations.

3 Architectural Overview

An initial overview of required steps for resolving all kinds of heterogeneities was shown in Fig. 1 above. In order to provide transformations independently of input (e.g., JSON) and output formats (e.g., OWL/XML), we propose a model-driven approach to bridging data model heterogeneity. In the following, the approach is detailed by means of a source JSON document, as extracted from a social network, and a target OWL/XML document, including the corresponding transformation of schemas. The proposed model-driven transformation process for resolving this data model heterogeneity is depicted in Fig. 2, and discussed in the following.

Meta-modeling layers and transformations.

Our approach anchors the transformations of schemas and instances along the four meta-modeling layers of Mof (Object Management Group 2011). The bottom layer (M0) describes actual instances (e.g., user profile instance data from Facebook in JSON or in an OWL ontology A-Box). These instances conform to models at the model layer (M1, e.g., the Facebook Graph API’s schema and JSON in general, or its representation as Facebook T-Box). In turn, these models conform to so-called meta-models (M2, e.g., JSON Schema as a language for defining the schemas of JSON documents or OWL as language for defining ontologies in the Semantic Web technology stack). Finally, meta-models are described in terms of a meta-meta-model (M3), such as Ecore 10.

In this four-layer representation, transformations for bridging technical spaces on a particular layer are always specified on the superior layer: thus, transformations on the M1 layer (e.g., from Facebook schema to a Facebook T-Box) are specified on the M2 layer, and transformations on the M0 layer (e.g., from Facebook instances to individuals in a Facebook A-Box) are specified on the M1 layer, as depicted in Fig. 2.

Schema and instance transformation.

For automatically transforming schemas, a transformation \( \text{trs} : L_{m2} \rightarrow L_{m2} \) has to be specified, for instance, from JSON Schema to OWL T-Box axioms, which allows to transform the corresponding schemas. For instance, a Facebook schema may be transformed to an according T-Box by executing the transformation specification, i.e., \( L_{m1s} = \text{trs}(L_{m1s}) \).

For actual instance data, existing approaches require specific instance transformation specifications \( \text{its} : L_{m1s} \rightarrow L_{m1s} \) between pairs of source \( L_{m1s} \) and target \( L_{m1s} \) models. Thus, to automatically transform Facebook instance data into Facebook A-Box axioms, a transformation specification between the Facebook Schema and the Facebook T-Box would be required, and for transforming data from LinkedIn

10Ecore is the realization of Mof in the Eclipse Modeling Framework (EMF) www.eclipse.org/modeling/emf

91
and Google+, another two specifications would be needed. We argue, that instead of multiple specific instance transformation specifications $tsi_1$, generic ones $tsi_g : L_{m1g} \rightarrow L_{m1g'}$ can be defined, if the source instances carry certain partial schema information when being interpreted in terms of a generic source language $L_{m1g}$, and the target language $L_{m1g'}$ supports DL and rule inference. Markup languages, such as JSON and XML, used in today’s online social network APIs, and OWL satisfy this requirement.

In the next section, we discuss our transformation approach for resolving data model heterogeneity between JSON Schema and OWL T-Box axioms on the model layer (M1), as well as between JSON and OWL A-Box axioms on the instance layer (M0). This transformation approach makes use of OWL inference capabilities to enable the specification of schema transformations on the meta-model layer, as well as instance transformations on the model layer, which replace manually created transformation specifications for each kind of data source.

4 JSON to OWL Transformation

In principle, our approach as depicted in Fig. 2 comprises two steps on the four-layer meta-modeling stack, transforming (i) JSON Schema to OWL T-Box axioms, and (ii) JSON instances to OWL A-Box axioms. Both these steps can be realized using model transformation techniques. The execution of a schema transformation specification, first, processes a source model (e.g., a Facebook schema deserialized from its textual representation) to create a corresponding target model (e.g., a Facebook T-Box, serializable into a textual representation). Second, when executed, a generic instance transformation specification (i.e., independent of Facebook Schema and T-Box) processes source instances (e.g., Facebook instances deserialized from JSON responses of Facebook) to create the target instances (e.g., a Facebook A-Box), which are serialized into a textual representation, such as OWL/XML. Finally, the result of the schema transformation execution is merged with the instance transformation execution result into a coherent Facebook ontology. The specifics of JSON and OWL, which are exploited in these generic transformations, are detailed below. The actual transformation specifications are given in Sect. 4.2.

4.1 JSON to OWL by Example

Let us consider user information from a social network (e.g., Facebook) extracted in JSON, as depicted in Fig. 3. This snippet, which is rather simple for the sake of understandability, shows a JSON object with a single property $\text{name}$, whose value is ‘Jane Doe’. The JSON object conforms to a simplified Facebook schema, defining that every $\text{User}$ is of type $\text{object}$ and comprises a property $\text{name}$.

Transformation of JSON Schema to OWL T-Box. In order to provide the Facebook schema in a T-Box (e.g., to support querying and reasoning), in a first transformation step, the schema is transformed into corresponding OWL T-Box axioms. These axioms define that $\text{User}$ is equivalent with the class of things that have a $\text{name}$ property of type $\text{String}$ (in OWL, this may be specified by the domain and range of a data property).

Transformation of JSON to OWL A-Box. In a second transformation step, the Facebook instance is transformed into corresponding OWL A-Box axioms, which again are specified in DL notation. As basic schema information, the format of the $\text{name}$ property’s value in our sample JSON snippet allows us to derive the property’s type: JSON distinguishes between string, number, boolean, object, and array. Further schema information, such as the concrete type of object (e.g., a person vs. an address), is not available in the instances. Anyhow, this is where OWL, implementing the family $\text{SROIQ}$ of description logic (Grau et al. 2008), is a perfect fit on the target side: description logic reasoners are specifically designed to classify objects according to their role assertions, and hence, are able to infer the schema information that is not explicitly present in JSON instances (e.g., given a sample T-Box axiom specified in description logic notation $\text{User} \equiv \exists \text{name}.\text{String}$, a description logic reasoner infers that anything with
at least one name is a user). Solutions to cope with potential ambiguities will be discussed in Section 4.4 below. As a result, the instance transformation specifications do not necessarily need information from a concrete model (e.g., a Facebook schema) for transformation. Thus, all JSON objects can be transformed into generic concept assertions of the kind Thing(a) (instead of specific ones, such as User(a)). In a similar manner, every value of a primitive property (e.g., 'Jane Doe') can be transformed into a concept assertion of the kind Literal. Finally, the connections between objects and the values of their primitive and complex properties (e.g., the fact that our sample user has the name ‘Jane Doe’) have to be transformed into role assertions (e.g., name(a, ‘Jane Doe’)). In case that a primitive or complex property allows an array of values, every array element must be represented with a corresponding role assertion.

Merging of OWL T-Box and A-Box. When being merged with the T-Box axioms, a description logic reasoner, such as HermiT\(^{11}\), infers concept assertions, as explained above. As a result, we can specify all instance transformations \(tsi : M_{m1y} \rightarrow M_{m1y}^{f}\) between JSON as source model and OWL A-Box axioms as target model in a generic manner. The specifications for both, generic schema and instance transformation, are detailed below.

### 4.2 Transformation Specifications

For specifying the actual schema and instance transformations, one may resort to existing (model) transformation languages, such as ATL (Jouault et al. 2008), QVT (Object Management Group 2009), or XSLT. Since existing tools for publishing social network data in RDF format use various transformation languages, we utilize our Mapping Operator language (MOps) (Wimmer et al. 2010b), which is designed as a suitable basis for creating transformation specifications in different kinds of transformation languages. Also, in our prototype, we use an executable implementation of MOps to perform the actual transformation. The basic building blocks of the MOps language for specifying transformations are so-called kernel MOps, which are composed to reusable higher-level transformations, denoted as composite MOps. Kernel MOps, their interplay, as well as the composition to composite MOps are explained below, as part of their application to schema and instance transformation specification. For a detailed description of kernel and composite MOps we refer to (Wimmer et al. 2010a,b).

---

11 hermit-reasoner.com

---

Schema transformation specification. The specification of schema transformations from JSON Schema to OWL T-Box using MOps is depicted in Fig. 4. Each MOp has input ports for accepting input on the left side and output ports for producing target objects on the right side. Ports are typed to classes (C) or attributes (A).

The JSON Schema meta-model is a subset of the JSON Schema Internet Draft\(^{12}\), which was selected for ease of presentation in a straightforward manner. For the OWL T-Box meta-model, we based on OMG’s Ontology Definition Metamodel\(^{13}\). Note, that for presentation purposes, the OWL T-Box meta-model was simplified, i.e., axioms for domain and range of properties are modeled as relationships (instead of subclasses of Axiom). In principle, every source element (Schema and Property) results in a target Declaration, as indicated by two Copier MOps in Fig. 4. The kind of declared entity depends on the type of the transformed source Schema: (i) complex schemas (type has value “object” or “array”) can best be represented as instances of Class in owl, while (ii) primitive schemas naturally become instances of Datatype. Since the topmost complex schema must be additionally transformed into an instance of Ontology (every OWL ontology is represented with one such instance), a composite MOp in terms of a horizontal partitioner is used. Such an HPartitioner comprises several CondCopiers, which restrict the output of a Copier to a subset satisfying a certain condition—in our case an ontology is only output for the topmost schema, and classes are only output for those schemas with type “array” or “object”. For transforming primitive schemas (i.e., simple types), we create an instance of Datatype for each distinct value of the type property in the source model (i.e., one datatype for “string”, one for “boolean”, and another one for “number”). Hence, the composite MOp ObjGenerator is used, which is depicted in white-box view, exposing the comprised kernel MOps. It includes (i) an A2C kernel MOp (transforms the value of an attribute of the source meta-model into a class instance of the target meta-model) for creating the instance of Datatype, and (ii) an A2A kernel MOp (copies an attribute value into another attribute value) for setting the IRI (Internationalized Resource Identifier) of the newly created datatype. Since there are dependencies between these kernel MOps (i.e., the IRI can only be set after the datatype has been created), the A2A MOp is additionally linked to the A2A MOp through a trace port (T), which provides context information about

---

\(^{12}\) tools.ietf.org/html/draft-zyp-json-schema-03

\(^{13}\) www.omg.org/spec/ODM/1.0
the produced output objects. Finally, instances of Property must either be transformed to instances of ObjectProperty (in case they reference a complex schema) or to DataProperty (in all other cases). Analogously to transforming instances of Schema, an HPartitioner can be used for that.

**Instance transformation specification.** Fig. 5 summarizes the instance transformations between JSON and OWL. The meta-models were built on basis of the same specifications as those for schema transformation.

As already discussed in the transformation example above, JSON objects are transformed in a straightforward manner to OWL individuals of unspecified type (i.e., Things). Hence, one Copier creates an Individual for each Object, while another one creates a ClassAssertion referring to the entity Thing. Node identifiers (iri) are generated, if present, from nested key properties (“id”, “key”), or, otherwise, from the filename (for root objects) or employing a simple heuristic, which uses the hash value of all nested members. Thereby, same objects can be matched (i.e., get equal identifiers), as long as they do not contain another layer of nested objects. In that case, distinct iris are generated.

Analogously to the schema transformation specification, on the instance level we distinguish between members of complex and of primitive type. Consequently, we utilize an HPartitioner to create an ObjectPropertyAssertion for members with complex values and a DataPropertyAssertion for members with primitive values. These assertions must reference the corresponding entities that were created during schema transformation (e.g., a “firstName” member must be transformed into an assertion of the corresponding “firstName” data property). Another Copier, again depicted in white-box view, creates new Entity instances from Member instances (the name of the member serves as the entity’s iri). During merging of T-Box and A-Box, iri equivalence ensures that the A-Box entities can be connected to their counterparts in the T-Box. Finally, every primitive value (Boolean, Number, and String) must be copied to an instance of Literal, but only if it was not created before (hence, the HMerger contains in fact three CondCopiers).

### 4.3 Implementation in EMF

As implementation platform, we chose the Eclipse Modeling Framework (EMF), including Xtext and Xtend for text (de-)serialization and transformation, due to its maturity and large community support. For implementing MOps and transformations in Xtend, meta-models in Ecore are required. The meta-models for JSON, JSON Schema, and OWL were generated from Xtext grammars according to the specifications introduced above. These Ecore meta-models are automatically translated by EMF into Java classes, which can then be used in Xtend. To switch to a different source technical space — in the past Facebook and the Twitter streaming API switched from XML to JSON — our implementation would only require Xtext grammars to generate the new meta-models for de-serialization to Ecore.

In order to utilize existing ontology tools and description logic reasoners, the Ecore representations of T-Box and A-Box resulting from applying the Xtend implementation of our MOps, then, must be serialized (e.g., as RDF/XML or OWL/XML) and merged. Again, Xtend is used for this purpose. For increased compatibility we implemented serializers for both, OWL/XML, e.g., for loading in Protégé, and RDF/XML, as required for Apache Jena.

### 4.4 Type Inference & Reasoning

Loading the generated files in Protégé enables the application of the included HermiT reasoner for type inference. Generic instance transformation specifications not taking into account schema information, however, may result in ambiguities during classification by a description logic reasoner. First, classes having the same mandatory, but different optional properties, cannot be matched unambiguously, in case that an object thereof is described in terms of the mandatory properties, only. Second, equally named and typed properties with different constraints in two different classes cannot be distinguished (e.g., a
class AustrianAddress with a 4-digit zip code vs. a GermanAddress with 5 digits).

To alleviate this problem, we propose to encode additional meta information from instance data in an owl T-Box — in a similar manner as previously shown for schema extraction (Kapsammer et al. 2012). For example, the property category of Facebook pages (a key concept in Facebook data, used universally) may be used to define equivalences with specific classes: Restaurant = Page ∩ category={restaurant}. Previously (Kapsammer et al. 2012), various heuristics for tackling this problem have been proposed: for example, IdFromValue infers a class name from property values (e.g., “type” or “category” in Facebook, and property “kind” in Google+), while IdFromReferenceName infers a class name from the names of references to nested individuals (e.g., used in LinkedIn). These different heuristics can be exploited during instance transformation as well. Therefore, each heuristic is encoded as a generic declarative Jena rule with accompanying built-ins for imperative computations. For instance, regarding connections in Facebook, the API’s JSON response, if requested, contains an object metainfo, containing an array connections, which in turn contains multiple properties, having URIs as values. These are links for other requests to receive further data, but they would actually resemble a connection between the response’s root object, and the root object of another JSON object (e.g., a list of friends). This LinkFromPatternFromValue strategy can be expressed as a Jena rule, thereby providing a direct edge between such objects (resulting in one edge per connection, instead of two intermediate nodes and three edges). As an alternative to built-ins, such definitions can be automatically generated by our previously proposed schema extraction process (Kapsammer et al. 2012) in terms of T-Box axioms, or as reasoning rules for semantic web rule reasoners (e.g., Jena inference rules).

5 Results & Evaluation

In this section, we evaluate our prototypical implementation by means of transforming data from Facebook, Google+, and LinkedIn to corresponding owl ontologies. Thereby, we will discuss several aspects: completeness, consistency, conciseness, as well as performance and scalability.

Evaluation Setup. In order to obtain comparable results, equal test user profiles were created in each of the selected social networks and extracted via their API. These profiles contain basic user information, jobs, education, as well as a connected friend for direct communication and interaction within a group. Note, that these data sets, since they were created manually, do not reflect the complete information available in real social network profiles. Nonetheless, as they were created in a consistent manner, they are suited for a first evaluation. (cf. (Kapsammer et al. 2012) for details on user profiles and generated schemas). The input data sets from Facebook, Google+, and LinkedIn as JSON files, as well as a simple introductory example, are available online17. Furthermore, the files include generated JSON Schema files, as explained by (Kapsammer et al. 2012). T-Box and A-Box in Ecore format, the merged serialization thereof (in RDF/XML & OWL/XML), as well as (for the three social networks) Jena rules and reasoning results.

Completeness. The completeness of the extracted data from social network APIs was already discussed previously (Kapsammer et al. 2012). The requirement of all information from the JSON input to be present in the output is fulfilled by the proposed transformation approach: all JSON objects, arrays, and simple types are transformed to owl (i.e., all input is present in output). This was evaluated by manually comparing the input to the generated instances, object properties, and datatype properties, on multiple samples from all four data sets.

Consistency. To evaluate the consistency of the transformations, we compared the outputs of multiple runs on the same input data. First, the more or less random serialization order of assertions is not a problem, since Protégé shows classes and individuals in alphabetical order anyway. Second, as discussed

---

17social-nexus.net/publications
Table 1: Description of input, output, and execution times—including count of files and individuals, file sizes (plain/zip-compressed), generated IDs and ID mismatches, as well as average execution times for transformations and reasoning.

<table>
<thead>
<tr>
<th>Input</th>
<th>Simple Ex.</th>
<th>Facebook</th>
<th>Google+</th>
<th>LinkedIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of input instance files (JSON)</td>
<td>1</td>
<td>192</td>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>Size of input files (JSON Schema + JSON) plain/compressed</td>
<td>1/1 kB</td>
<td>399/128 kB</td>
<td>31/12 kB</td>
<td>41 (14) kB</td>
</tr>
<tr>
<td>Output</td>
<td>Simple Ex.</td>
<td>Facebook</td>
<td>Google+</td>
<td>LinkedIn</td>
</tr>
<tr>
<td>Number of individuals (in A-Box)</td>
<td>4</td>
<td>1549</td>
<td>104</td>
<td>260</td>
</tr>
<tr>
<td>Size of output files (T-Box, A-Box &amp; inferred triples in RDF/XML)</td>
<td>13/2 kB</td>
<td>626/63 kB</td>
<td>58/6 kB</td>
<td>113/9 kB</td>
</tr>
<tr>
<td>Number of individuals without unique ID from JSON input</td>
<td>2</td>
<td>1078</td>
<td>64</td>
<td>187</td>
</tr>
<tr>
<td>Mismatches of generated IRI for objects with equal content</td>
<td>0</td>
<td>53</td>
<td>4</td>
<td>25</td>
</tr>
</tbody>
</table>

| Transformation & Reasoning Time | Simple Ex. | Facebook | Google+ | LinkedIn |
| Sum of transformation & reasoning time | 196 ms | 8621 ms | 2198 ms | 3186 ms |
| JSON Schema to OWL T-Box Ecore | 45 ms | 107 ms | 64 ms | 67 ms |
| JSON to OWL A-Box Ecore | 62 ms | 2541 ms | 282 ms | 559 ms |
| OWL Ecore to RDF/XML & OWL/XML | 90 ms | 2322 ms | 397 ms | 774 ms |
| Jena Rule Reasoning on RDF/XML | - | 3651 ms | 1455 ms | 1785 ms |

earlier, for individuals without a unique ID from the JSON input, an identifier has to be generated. These generated IRIs may differ between runs, but are always consistent within a generated ontology. In this context, Table 1 also counts mismatches of generated IRIs for objects with equal content (i.e., the heuristic fails, if JSON objects contain nested objects, as discussed in Sect. 4.2).

Conciseness. Comparing the file size for JSON input and OWL output, Table 1 shows that plain RDF/XML files are larger than the JSON counterparts. Zip-compression, however, works much more efficiently for the RDF/XML format.

Regarding the conciseness of transformed output itself, we compare the generated serialization to the minimal representation of the same facts. Such a minimal OWL A-Box would contain one assertion for each individual, datatype property, and object property. In this sense, the generated RDF/XML is not minimal, as we assert all individuals as Things, and the reasoner adds inferred assertions with more specific types. For properties, however, the transformation is minimal. Concerning the JSON tree structure, in which data is provided by social network APIs, the length of paths is crucial for navigation. Our implementation converts these trees, being in fact a special case of graphs, to tree-like ontology graphs, which then allow the introduction of shortcuts (e.g., for Facebook connections, as discussed in Sect. 4.4). On one hand, these shortcuts represent additional assertions, but, on the other hand, materializing these shortcuts enable faster queries that are also easier to express.

Performance & Scalability. To evaluate the time complexity of our approach, we measured the execution times for the different data sets in our prototype, as shown in Table 1. From these measurements we can observe that the size of the JSON Schema correlates with its transformation time to Ecore. The same applies for the transformation of JSON instances to Ecore. Concerning the Jena inference step, clearly the reasoning takes most of the time in the overall process, with the benefit of being able to infer information that is not present in the source data explicitly. Note, that the times in the above table include file I/O. In a separate run without those JSON files from the Facebook input, which contain almost no members (i.e., empty objects and arrays), the transformation of instances to Ecore was sped up by 37%, with remaining 92% of individuals within one third of files. Thus, to make the transformation more efficient, all steps may be integrated into a single application with reduced file system access for input and intermediate files. Concerning scalability, the average transformation times for larger inputs grow, from a certain point, linearly (transformation & serialization), and exponentially for the reasoning step. However, using specific rules that do not require imperative computations (cf. Sect. 4.4), the magnitude would probably be reduced. On average, for 62,000 individuals plus properties, the transformation to Ecore took 37 seconds, the Jena inferring took 184 seconds.

Finally, memory complexity was measured in terms of RAM consumption. Whereas for transformation of these 62,000 individuals plus properties the Java process consumed 1025 MB, growing linearly, for the reasoning it peaked at 335 MB, almost constantly (i.e., almost no growth).

6 Conclusion and Future Work

We presented an approach for model driven transformation of schemas and instances between different technical spaces. Our method requires the transformation specification to be done only once, for instance, from JSON Schema to OWL T-Box axioms. This specification can then be executed for different data sources, such as different social networks providing JSON data via their APIs. Neglecting semantics, JSON’s tree-like structure can be transformed to OWL in a straightforward manner (e.g., JSON slots of simple datatype as datatype properties, complex types and arrays as object properties). To go beyond syntactic equivalence, and to consider semantics of APIs, some configuration was required, which obviously depends on the data source. For instance, semantic equivalence from JSON slots is not generally possible, but in Facebook ID slots can be used to define semantic equivalence.

In the evaluation section we applied our approach to comparable user profiles from Facebook, Google+, and LinkedIn. Not surprisingly, time and memory complexity were relatively high (especially for reasoning and for larger user profiles), but clearly there is potential for optimizations, and an evaluation on extensive data sets would be interesting for future work.
Our focus was on the pre-requisite steps for data integration and consolidation from different social networks: the goal was to overcome data model heterogeneity, in order to facilitate structural and semantic integration later on. In contrast to related transformation approaches, using model driven architectures allows to build graphical editors and to cope with evolution, for instance when APIs change. Also, they enable source/target formats exchange without influencing transformation rules, and platform-independent MOps allow replacing the transformation platform.

**Generalization to arbitrary source and target models.** For generalizing the presented approach to technical spaces other than JSON as source and OWL as target, several modifications need to be taken into account. Without the reasoning capabilities of OWL and Jena rules, instances need to be loaded according to their concrete source schema (instead of some generic meta-model), therefore, preventing generic instance transformations that are independent of source models. As a consequence, firstly, dedicated (de-)serializers for every single source and target schema are required, and, secondly, the generic instance transformation specification (specified on the model layer) must be replaced with be replaced with specific ones for each social network schema. In order to automate this whole process, the deserializers for source schemas should be generated automatically. Furthermore, the instance transformation specifications on the model layer are foreseen to be created automatically as artifacts as well—just like the target model classes are generated—during the execution of the transformation from source to target model (specified on the meta-model layer). This means, a sole transformation specification on the meta-model layer may result in potentially many transformation specifications on the model layer. However, as a result of the limitations of current model engineering software frameworks (specifically, the fact that Eclipse modeling spans three of the four meta-modeling layers), the models created during schema transformation must be lifted to the meta-model layer first. This means, that instances in the schema transformation specifications must become models in the instance transformation specification.

**Semantic integration of schemas.** Having solved data model heterogeneity between different social networks by transformation to OWL, instance based schema matching tools, such as COMA++ (Massmann et al. 2011), may be used to support humans in defining semantic correspondences. Such tool support is especially helpful for large or unknown schemas. However, unlike transformations for resolving technical heterogeneity, which were shown in this paper to be specifiable in a generic manner, transformations resolving semantic heterogeneity still need manual intervention. Therefore, we can resort to Semantic Web technologies to integrate the transformed user models in OWL/XML, for instance, by defining equivalences between the OWL classes User from Facebook and Person from FOAF. Again, employing reasoners allows to retrieve instances materialized as Facebook A-Box from queries using the FOAF vocabulary.

**Source Schema Evolution.** Once such mappings are specified, an especially interesting question therefore is, how to automate or support evolution of these transformations. A first idea in this direction is the design of a meta-model of possible schema changes in EMF, including generic operations to propagate changes to dependent artifacts, such as queries and integration rules.

**References**


Rowe, M. & Ciravegna, F. (2008), Getting to ‘Me - Exporting Semantic Social Network Information from Facebook, in ‘1st Social Data on the Web workshop (SDoW2008)’.


Author Index

Achuthan, Narasimaha R, 69
Affleck, Amy, 69
Benda, Soběslav, 31
Brown, Ross, 51
Ferrarotti, Flavio, iii
Grossmann, Georg, iii
Johnson, Daniel, 51
Kapsammer, Elisabeth, 89
Klímek, Jakub, 31
Krishna, Aneesh, 69
Kusel, Angelika, 89
Lechner, Stephan, 89
Link, Sebastian, 41
Looser, Dominic, 61
Low, W. Z., 79
Ma, Hui, 61
Malý, Jakub, 21
Memari, Mozghan, 41
Mitsch, Stefan, 89
Nauta, W., 79
Nečaský, Martin, 21, 31
Neumayr, Bernd, 3
Poppe, Erik, 51
Pröll, Birgit, 89
Recker, Jan, 51
Retschitzegger, Werner, 89
Schönböck, Johannes, 89
Schewe, Klaus-Dieter, 61
Schrefl, Michael, 3
Schwinger, Wieland, 89
Stumptner, Markus, 3
Wimmer, Manuel, 89
Wischenbart, Martin, 89
Wynn, M. T., 79
Recent Volumes in the CRPIT Series

**Volume 113 - Computer Science 2011**

**Volume 114 - Computing Education 2011**

**Volume 115 - Database Technologies 2011**

**Volume 116 - Information Security 2011**

**Volume 117 - User Interfaces 2011**

**Volume 118 - Parallel and Distributed Computing 2011**

**Volume 119 - Theory of Computing 2011**

**Volume 120 - Health Informatics and Knowledge Management 2011**

**Volume 121 - Data Mining and Analytics 2011**
Edited by Peter Vampile, University of Ballarat, Australia, Andrew Ryan, University of Ballarat, Australia, Kok-Leong Ong, Deakin University, Australia, Peter Christen, Australian National University, Australia and Paul J. Kennedy, University of Technology, Sydney, Australia. December 2011. 978-1-921770-02-9. Contains the proceedings of the Ninth Australasian Data Mining Conference (AusDM’11), Ballarat, Australia, 1-2 December 2011.

**Volume 122 - Computer Science 2012**
Edited by Mark Reynolds, The University of Western Australia, Australia and Bruce Thomas, University of South Australia. January 2012. 978-1-921770-03-6. Contains the proceedings of the Thirty-Fifth Australasian Computer Science Conference (ACSC 2012), Melbourne, Australia, 30 January – 3 February 2012.

**Volume 123 - Computing Education 2012**

**Volume 124 - Database Technologies 2012**

**Volume 125 - Information Security 2012**

**Volume 126 - User Interfaces 2012**
Edited by Haifeng Shen, Flinders University, Australia and Peter Christen, Australian National University, Australia. December 2011. 978-1-920682-97-2. Contains the proceedings of the Thirteenth Australasian User Interface Conference (AUIC2012), Melbourne, Australia, 30 January – 3 February 2012.

**Volume 127 - Parallel and Distributed Computing 2012**

**Volume 128 - Theory of Computing 2012**

**Volume 129 - Health Informatics and Knowledge Management 2012**
Edited by Keryn Butler-Henderson, Curtin University, Australia and Kathleen Gray, University of Melbourne, Australia. January 2012. 978-1-921770-10-4. Contains the proceedings of the Fifth Australasian Workshop on Health Informatics and Knowledge Management (HIKM 2012), Melbourne, Australia, 30 January – 3 February 2012.

**Volume 130 - Conceptual Modelling 2012**

**Volume 134 - Data Mining and Analytics 2012**
Edited by Yanchang Zhao, Department of Immigration and Citizenship, Australia, Junqing Li, University of South Australia, Paul J. Kennedy, University of Technology, Sydney, Australia and Peter Christen, Australian National University, Australia. December 2012. 978-1-921770-14-2. Contains the proceedings of the Tenth Australasian Data Mining Conference (AusDM’12), Sydney, Australia, 5-7 December 2012.