

CONFERENCES IN RESEARCH AND PRACTICE IN
INFORMATION TECHNOLOGY

VOLUME 53

CONCEPTUAL MODELLING 2006

AUSTRALIAN COMPUTER SCIENCE COMMUNICATIONS, VOLUME 28, NUMBER 6.



AUSTRALIAN
COMPUTER
SOCIETY



CO_{mputing}
R_{esearch}
& E_{ducation}

CONCEPTUAL MODELLING 2006

Proceedings of the 3rd Asia-Pacific
Conference on Conceptual Modelling (APCCM2006),
Hobart, Tasmania, Australia, 16-19 January 2006

Markus Stumptner, Sven Hartmann and Yasushi Kiyoki,
Eds.

Volume 53 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.

**Proceedings of the Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006),
Hobart, Tasmania, Australia, 16-19 January 2006**

Conferences in Research and Practice in Information Technology, Volume 53.

Copyright ©2006, Australian Computer Society. Reproduction for academic, not-for profit purposes permitted provided the copyright text at the foot of the first page of each paper is included.

Editors:

Markus Stumptner
School of Computer and Information Science
University of South Australia
Mawson Lakes 5095
South Australia
Email: mst@cs.unisa.edu.au

Sven Hartmann
Department of Information Systems
Massey University
Palmerston North
New Zealand
Email: s.hartmann@massey.ac.nz

Yasushi Kiyoki
Department of Environmental Information
Keio University
Tokyo
Japan
Email: kiyoki@mdl.sfc.keio.ac.jp

Series Editor: John F. Roddick,
Conferences in Research and Practice in Information Technology
Flinders University,
PO Box 2100, Adelaide 5001
South Australia.
crpit@infoeng.flinders.edu.au

Publisher: Australian Computer Society Inc.
PO Box Q534, QVB Post Office
Sydney 1230
New South Wales
Australia.

Conferences in Research and Practice in Information Technology, Volume 53.
ISSN 1445-1336.
ISBN 1-920-68235-X.

Printed, December 2005 by Flinders Press, PO Box 2100, Bedford Park, SA 5042, South Australia.
Cover Design by Modern Planet Design, (08) 8340 1361.

The *Conferences in Research and Practice in Information Technology* series aims to disseminate the results of peer-reviewed research in all areas of Information Technology. Further details can be found at <http://crpit.com/>.

Table of Contents

Proceedings of the Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Tasmania, Australia, 16-19 January 2006

Preface	vii
Programme Committee	ix
Organising Committee	x
CORE - Computing Research and Education	xi
ACSW Conferences and the Australian Computer Science Communications	xii
ACSW and APCCM 2006 Sponsors	xv

Keynote

Conceptual Requirements Modeling - a Contribution to XNP (eXtreme Non Programming)	3
<i>Heinrich C. Mayr</i>	

Invited Papers

ServiceMosaic Project: Modeling, Analysis and Management of Web Services Interactions	7
<i>Boualem Benatallah and Hamid Reza Motahari-Nezhad</i>	
Postmodern Prospects for Conceptual Modelling	11
<i>James Noble and Robert Biddle</i>	
Network Data Mining: Methods and Techniques for Discovering Deep Linkage between Attributes ...	21
<i>John Galloway and Simeon J. Simoff</i>	

Contributed Papers

The Formal Semantics of the TimeER Model	35
<i>Heidi Gregersen</i>	
A Conceptual Solution for Representing Time in Data Warehouse Dimensions	45
<i>Elzbieta Malinowski and Esteban Zimányi</i>	
Visualization of Music Impression in Facial Expression to Represent Emotion	55
<i>Takafumi Nakanishi and Takashi Kitagawa</i>	
Modelling Human Perception to Leverage the Reuse of Concepts across the Multi-sensory Design Space	65
<i>Keith V. Nesbitt</i>	
Process Modelling: The Deontic Way	75
<i>Vineet Padmanabhan, Guido Governatori, Shazia Sadiq, Robert Colomb and Antonino Rotolo</i>	

Defining and Implementing Domains with Multiple Types using Mesodata Modelling Techniques	85
<i>Sally Rice, John F. Roddick and Denise de Vries</i>	
On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling	95
<i>Nick Russell, Wil M.P. van der Aalst, Arthur H.M. ter Hofstede and Petia Wohed</i>	
Component-Driven Engineering of Database Applications	105
<i>Klaus-Dieter Schewe and Bernhard Thalheim</i>	
Supporting Virtual Organisation Alliances with Relative Workflows	115
<i>Xiaohui Zhao, Chengfei Liu and Yun Yang</i>	
Author Index	125

Preface

This volume contains the papers presented at the *Third Asia-Pacific Conference on Conceptual Modelling (APCCM 2006)* which was held in Hobart, Australia from January 16 to 19, 2006 as part of the Australasian Computer Science Week (ACSW 2006). On behalf of the programme committee we commend these papers to you and hope you find them useful.

The Asia-Pacific Conference on Conceptual Modelling 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. Embedding APCCM into the Australasian Computer Science Week gives our conference participants the opportunity to attend also other conferences, including the 17th Australasian Database Conference (ADC 2006).

The scope of APCCM 2006 includes areas such as:

- business, enterprise and process modelling
- concepts, concept theories and ontologies
- conceptual modelling for ...
 - decision support and expert systems
 - digital libraries, mobile information systems and web-based systems
 - e-business, e-commerce and e-banking systems
 - knowledge management systems
 - semi-structured data and XML
 - spatial, temporal and biological data
 - user interfaces and user participation
- conceptual modelling quality
- 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
- philosophical, mathematical and linguistic foundations of conceptual models
- reuse, reverse engineering and reengineering
- the Semantic Web
- software engineering and tools for information systems development

Following a call for papers, which yielded a record 40 abstracts and 34 full paper submissions, there was a rigorous refereeing process that saw all papers refereed by at least three international experts. The nine papers judged best by the programme committee were accepted and are included in this volume.

In addition, the programme committee invited Heinrich Mayr to present the keynote lecture on *Conceptual Requirements Modeling - a Contribution to NP (eXtreme Non Programming)*, while three invited speakers kindly agreed to give talks on their work: Boualem Benatallah on *ServiceMosaic Project: Modeling, Analysis and Management of Web Services Interactions*, James Noble on *Postmodern Prospects for Conceptual Modelling*, and Simeon Simoff on *Network Data Mining: Methods and Techniques for Discovering Deep Linkage between Attributes*.

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 programme committee and the additional reviewers for their timely expertise in carefully reviewing the papers, and to Markus Kirchberg for his willingness to continue his excellent work as APCCM Publicity Chair. Finally, we wish to express our appreciation to the local organisers at the University of Tasmania for the wonderful days in Hobart.

Markus Stumptner

(University of South Australia)

Sven Hartmann

(Massey University, New Zealand)

Yasushi Kiyoki

(Keio University, Japan)

APCCM 2006 Programme Chairs

January 2006

Programme Committee

Chairs

Markus Stumptner, University of South Australia
Sven Hartmann, Massey University
Yasushi Kiyoki, Keio University

Members

Boualem Benatallah, University of New South Wales
Xing Chen, Kanagawa Institute of Technology
Gill Dobbie, University of Auckland
Cesar Gonzalez-Perez, University of Technology, Sydney
John Grundy, University of Auckland
Kathleen Hornsby, University of Maine
Yoshihide Hosokawa, Nagoya Institute of Technology
Sebastian Link, Massey University
Jixue Liu, University of South Australia
Mukesh Mohania, IBM India Research Lab
Maria Orlowska, University of Queensland
John Roddick, Flinders University, Adelaide
Michael Rosemann, Queensland University of Technology
Gunter Saake, University of Magdeburg
Michael Schrefl, University of Linz
Nigel Stanger, University of Otago
Bernhard Thalheim, University of Kiel
Naofumi Yoshida, Keio University
Jeffrey Xu Yu, Chinese University of Hong Kong
Kouji Zettsu, NICT, Japan
Yanchun Zhang, Victoria University
Jane Zhao, Massey University

Publicity Chair

Markus Kirchberg, Massey University

Additional Reviewers

Georg Grossmann, University of South Australia
Markus Kirchberg, Massey University
Hui Ma, Massey University
Jiangang Ma, Victoria University, Melbourne
Jun Shen, University of South Australia
Jing Wang, Massey University
Guandong Xu, Victoria University, Melbourne
Sergiy Zlatkin, Massey University

Organising Committee

Welcome

On behalf of the Tasmanian Organising Committee of ACSW2006 I would like to welcome all the delegates to the conferences of this busy and interesting week, in particular those coming from overseas.

The location of the various conferences and other events at the Wrest Point Hotel allows delegates to move quickly from event to event, and to easily and comfortably gather in groups for those conversations and interactions that are so important for the exchange of ideas and the promotion of cooperation, not to mention social pleasure.

We trust you will have a thoroughly enjoyable time.

Professor Young Ju Choi
Chair, Organising Committee
January, 2006

General Chair

Professor Young Ju Choi, School of Computing, University of Tasmania, Australia

Organising Committee Members

Ms Nicole Clark
Dr Julian Dermoudy
Mr Tony Gray
Mr Neville Holmes
Mr Ian McMahon
Ms Julia Mollison
Professor Arthur Sale
Ms Soon-ja Yeom

CORE - Computing Research and Education

CORE welcomes all delegates to ACSW2006 in Hobart.

ACSW, the Australasian Computer Science Week continues to grow with new conferences becoming entrenched in the week. As the premier annual Computer Science event in Australia and New Zealand, it provides an unparalleled opportunity for the wide community of Computer Science academics and researchers to meet, network, promote IT research and be exposed to the latest research in other areas of IT. The research presented at each conference is of the highest standard and essential for the growth and future of our region, in an ever more competitive world.

CORE is expanding its awards. The Distinguished Service Award first offered in late 2004 will be offered every second year and next at the 2007 Conference. Along with the Chris Wallace Research Award, we are offering an annual teaching award for the first time.

CORE has continued to play a part in the Federation of Australian Scientific and Technological Societies and by participating in events such as Science Meets Parliament, CORE is becoming recognised by the wider community and will continue to do so. A major contribution from many members in 2005 was a submission to the RQF Forum with some of our ideas appearing in the draft. CORE and members of the Executive have also been interviewed as representatives of the Computer Science community for several other Government and industry inquiries and initiatives.

Thank you all for your contributions in 2005 and we look forward to an exciting 2006.

Jenny Edwards
President, Computing Research and Education
January, 2006

ACSW Conferences and the Australian Computer Science Communications

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.

- 2008. Communications Volume Number 30. Proposed Host and Venue - University of Wollongong, NSW.
- 2007. Volume 29. Host and Venue - University of Ballarat, VIC.
- 2006. **Volume 28. Host and Venue - University of Tasmania, TAS.**
- 2005. Volume 27. Host - University of Newcastle, NSW. APBC held separately from 2005.
- 2004. Volume 26. Host and Venue - University of Otago, Dunedin, New Zealand. First running of APCCM.
- 2003. Volume 25. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Adelaide Convention Centre, Adelaide, SA. First running of APBC. Incorporation of ACE. ACSAC held separately from 2003.
- 2002. Volume 24. Host and Venue - Monash University, Melbourne, VIC.
- 2001. Volume 23. Hosts - Bond University and Griffith University (Gold Coast). Venue - Gold Coast, QLD.
- 2000. Volume 22. Hosts - Australian National University and University of Canberra. Venue - ANU, Canberra, ACT. First running of AUIC.
- 1999. Volume 21. Host and Venue - University of Auckland, New Zealand.
- 1998. Volume 20. Hosts - University of Western Australia, Murdoch University, Edith Cowan University and Curtin University. Venue - Perth, WA.
- 1997. Volume 19. Hosts - Macquarie University and University of Technology, Sydney. Venue - Sydney, NSW. ADC held with DASFAA (rather than ACSW) in 1997.
- 1996. Volume 18. Host - University of Melbourne and RMIT University. Venue - Melbourne, Australia. CATS joins ACSW.
- 1995. Volume 17. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Glenelg, SA.
- 1994. Volume 16. Host and Venue - University of Canterbury, Christchurch, New Zealand. CATS run for the first time separately in Sydney.
- 1993. Volume 15. Hosts - Griffith University and Queensland University of Technology. Venue - Nathan, QLD.
- 1992. Volume 14. Host and Venue - University of Tasmania, TAS. (ADC held separately at La Trobe University).
- 1991. Volume 13. Host and Venue - University of New South Wales, NSW.
- 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.
- 1988. Volume 10. Host and Venue - University of Queensland, QLD.
- 1987. Volume 9. Host and Venue - Deakin University, VIC.
- 1986. Volume 8. Host and Venue - Australian National University, Canberra, ACT.
- 1985. Volume 7. Hosts - University of Melbourne and Monash University. Venue - Melbourne, VIC.
- 1984. Volume 6. Host and Venue - University of Adelaide, SA.
- 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

- ACE.** Australian/Australasian Conference on Computing Education.
- ACSAC.** Asia-Pacific Computer Systems Architecture Conference (previously Australian Computer Architecture Conference (ACAC)).
- ACSC.** Australian/Australasian Computer Science Conference.
- ACSW.** Australian/Australasian Computer Science Week.
- ADC.** Australian/Australasian Database Conference.
- AISW.** Australasian Information Security Workshop.
- APBC.** Asia-Pacific Bioinformatics Conference.
- APCCM.** Asia-Pacific Conference on Conceptual Modelling.
- AUIC.** Australian/Australasian User Interface Conference.
- AusGrid.** Australasian Workshop on Grid Computing and e-Research.
- CATS.** Computing - The Australian/Australasian Theory Symposium.

Note that various name changes have occurred, most notably the change of the names of conferences to reflect a wider geographical area.

ACSW and APCCM 2006 Sponsors

We wish to thank the following sponsors for their contribution towards this conference. For an up-to-date overview of sponsors of ACSW 2006 and APCCM 2006, please see <http://www.comp.utas.edu.au/acsw06/>.



University of Tasmania, Australia



AUSTRALIAN
COMPUTER
SOCIETY

Australian Computer Society

CO_{mputing}
R_{esearch}
& E_{ducation}

CORE - Computing Research and Education



University of
South Australia

School of Computer and Information Science



Massey University, New Zealand

Keio University
A TRADITION OF EXCELLENCE

Keio University, Japan

KEYNOTE

Conceptual Requirements Modeling – a Contribution to XNP (eXtreme Non Programming)

Heinrich C. Mayr, IWAS - University of Klagenfurt, AUSTRIA

Despite numerous attempts, software and information system development suffers substantially from lacks in quality and effectiveness. E.g., studies show that in spite of all existing sophisticated software tools a high number of software and IS projects still fail to meet the “natural” requirements related to such projects: to stay in budget, to be finished in time, and to fulfill the application’s and the users’ needs.

To overcome these problems, increasingly techniques and tools are proposed and developed that focus on modeling instead of programming and that support the transformation of conceptual models / schemas into running software. Thus, as a logical consequence, it is also worthwhile and appropriate to investigate, to which extent the modeling process itself might be supported by automatic means. This is an important issue since even today the majority of failing software projects does so because of insufficiently elicited and/or validated business owner (stakeholder) requirements.

The talk outlines the actual state of the art in requirements modeling and presents the KCPM (Klagenfurt Conceptual Predesign Model) approach to extract models from natural language requirements specifications. This approach comes with a lean intermediate conceptual modeling language that is specifically oriented at the end-user’s capability in abstracting from real world observations and in understanding models of a complex reality. Specific linguistic categorization and semantic interpretation mechanisms support the creation of KCPM Schemes. Heuristic mapping rules are used by tools that transform KCPM schemes into traditional conceptual schemes like UML class diagrams, state charts, activity diagrams and others.

INVITED PAPERS

ServiceMosaic Project: Modeling, Analysis and Management of Web Services Interactions

Boualem Benatallah¹, Hamid Reza Motahari-Nezhad^{1,2}

¹University of New South Wales (UNSW)
NSW 2052, Australia

²National ACT of Australia (NICTA)
NSW 1430, Australia

{Boualem,hamidm}@cse.unsw.edu.au

Abstract

This paper provides an overview of ServiceMosaic, which is a platform for model-driven analysis and management of service interactions. In particular, in this paper, we focus on business protocols modelling and analysis by providing operators for compatibility and replaceability checking of business protocols, and model-driven adapter development for business protocols.

Keywords: Web Services, Business Protocols, Compatibility, Replaceability, Adaptation.

1 Introduction

Application integration has been one of the main drivers of software market into the new millennium. The main benefits that Web services bring to integration are (i) support for loosely coupled and decentralized interactions and (ii) standardization at different levels such as message format (XML), interface definition language (WSDL), and transport mechanism (SOAP, typically over HTTP) (Papazoglou and Georgakopoulos 2003, Alonso et. Al 2004). Standardization helps reduce the costs of application integration, which are to a large extent due to the fact that different interacting entities have different interfaces, speak different communication protocols, and support different data formats and interaction models.

Although Web services provide abstractions to simplify the integration at lower levels of the interaction stacks (e.g., data syntax and communication protocols), where many of the issues have already been identified or even solved, they have not (yet) contributed to simplify integration at higher abstraction levels (e.g., data/message types and business-level interaction protocols). In this paper, focus on integration issues at the business protocols level. Generally stated, a business protocol specifies the ordering constraints on the message exchanges, which are allowed by the service in the interactions with other services. We developed a model-driven framework, called *ServiceMosaic*, for modeling,

analysis and management of Web service interactions. We provide an overview of this work. We focus on business protocols and refer to other papers for detailed descriptions of the different concepts and techniques developed in this project.

2 Analysis and Management of Web Service Protocols

In (Benatallah, Casati, and Toumani, 2005), we have developed a protocol algebra and protocol management operators that are targeted at three main types of analysis, which we believe to be essential to Web service analysis and management: *Compatibility* (i.e., assessing if two services can interoperate correctly), *replaceability* (i.e., verifying whether two different protocols can support the same set of conversations), and *consistency* (i.e., verifying whether the implementation of a service can support the declared protocol definition).

2.1 Modelling Business Protocols

Several languages exist for describing Web service protocols (e.g., WS-BPEL, WSCI, WS-CDL) (Papazoglou and Georgakopoulos 2003, Alonso et. al 2004). These languages are concerned more with implementation aspects than specification of protocol properties. They are not suitable for automating activities such as protocol compatibility and compliance analysis. Our framework features a simple, high level but expressive enough model to represent features and abstractions that are useful and needed in practice (Benatallah, Casati, and Toumani, 2004). This model builds upon the traditional state-machine formalism to represent message choreography constraints and extends it to cater for relevant protocol abstractions such as temporal constraints or the implications and the effects of service invocations from requester perspective.

We model a service business protocol (protocol for short) as a non-deterministic finite state machine, where the states represent the different phases that a service may go through during its interaction with a requestor. Transitions are triggered by messages sent by the requestor to the provider or vice versa (hence, transitions are labeled with either input or output messages). A message corresponds to the invocation of a service operation or to its reply. Each service may be simultaneously involved in several message exchanges (conversations) with different clients, and therefore can be characterized by multiple concurrent instantiations of

Copyright (c) 2006, Australian Computer Society, Inc. This paper appeared at the *third Asia-Pacific Conference on Conceptual Modelling (APCCM2006)*, Hobart, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 53. Markus Stumptner, Sven Hartmann, and Yasushi Kiyoki, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

the protocol state machine. The purpose of the protocol is essentially to specify the set of conversations that are supported by the service.

The proposed model caters also for the modeling of transactional implications of operation invocations (Benatallah, Casati, and Toumani, 2004). We distinguish the following types of transitions of a protocol state machine. *Effect-less transitions* have no effect from the client's perspective. *Compensatable transitions* denote transitions whose effect can be cancelled. *Definite transitions* denote transitions whose transactional effects are permanent. *Resource-locking transitions* lock certain resources for the requester for a period of time. Temporal properties can also be specified by tagging transitions with a time interval, with the meaning that the transition is fired as the interval expires, hence leading the state machine to a new state from which previously invoked operations are not enabled any more (Benatallah, Casati, Ponge, and Toumani, 2005).

2.2 Compatibility Analysis

Compatibility analysis is concerned with verifying whether two services can interoperate. It is necessary for static and dynamic binding, and it also helps in evolution, since it helps verifying that a modified client can still interact as desired with a certain service. More precisely, we identified in two classes of compatibility:

- *Partial compatibility*: A protocol P_x is partially compatible with another protocol P_y if there are some executions of P_x that can interoperate with P_y , i.e., if there is at least one possible conversation that can take place among a service supporting P_x and a service supporting P_y .
- *Full compatibility*: a protocol P_x is fully compatible with another protocol P_y if all the executions of P_x can interoperate with P_y , i.e., any conversation that can be generated by P_x is understood by P_y .

These notions of compatibility are very useful in the context of Web services. For example, it does not make sense to have interactions with services for which there is no (partial or total) compatibility, as no meaningful conversation can be carried on. Furthermore, if there is only partial compatibility, the developer and the Web service middleware need to be aware of this, as the service will not be able to exploit its full capabilities when interacting with the partially compatible ones. When compatibility is partial, developers will likely want to know which conversations are allowed and which are not. We have developed an operator for compatibility checking that takes two protocols as input and returns the conversations that can take place between two services supporting these protocols (Benatallah, Casati, and Toumani, 2005).

2.3 Replaceability Analysis

Replaceability analysis helps identify if two protocols are equivalent in terms of conversations that they can support, both in general and when interacting with a

certain client. This can be for example used to determine whether a new version of a service (protocol) can support the same conversations as the previous one, or whether a newly defined service can support the conversations mandated by a given standard specification. We identified several replaceability classes, which provide basic building blocks for analyzing the commonalities and differences between service protocols:

Protocol equivalence. Two business protocols P_x and P_y are equivalent if they support the same set of conversations. Any conversation that is legal (i.e., does not result in errors) according to P_x will also be legal according to P_y , and vice versa. This means that the two protocols can be interchangeably used in any context and the change is transparent to clients.

Protocol subsumption. A protocol P_y is subsumed by another protocol P_x if P_x supports at least all the conversations that P_y supports. Hence, protocol P_x can be transparently used instead of P_y but the opposite is not necessarily true.

Protocol equivalence and subsumption with respect to a client protocol. It may be important to understand if a service can be used to replace another one when interacting with a certain client. This leads to a weaker definition of replaceability: a protocol P_x can replace another protocol P_y with respect to a client protocol P_c if every legal conversation between P_y and P_c is also a legal conversation between P_x and P_c . In this case, P_x can replace P_y to interact with P_c .

As for compatibility, the above discussion emphasizes the need for operators to analyze equivalence, subsumption, and different notions of replaceability. There is also the need for understanding, when two protocols are not equivalent, which conversations can be handled by both and which cannot. This leads to providing operators to determine *intersection* and *difference* among protocols, among others, to identify which conversations can and cannot be supported when a service is used in place of another (see (Benatallah, Casati, and Toumani, 2005) for details).

3 Model-driven Adapter Development for Web Service Protocols

When two services are not fully compatible or equivalent, we may consider adapting them. We classify the need for adaptation in Web services in two basic categories: adaptation for compatibility and for replaceability. The first category refers to wrapping a Web service so that it can interact with another service. It is needed when two services are functionality-wise compatible, but with incompatible interfaces or protocol specifications. The second category refers to modifying a Web service so that it becomes compliant with (i.e., can be used to replace) another service.

We take the view that although concrete adapter specifications are application-specific, in many cases it is possible to capture in a generic way the type of differences among protocol and the way to resolve them

into what we call *mismatch patterns*. Indeed, we have analyzed interfaces and protocols to identify the most typical differences and for these we have specified the corresponding mismatch patterns (Benatallah, Casati, Grigori, Motahari-Nezhad, and Toumani, 2005). This is akin to detecting structural and semantic differences in data mappings (Rahm and Bernstein, 2001). Each mismatch pattern includes an *adapter template* to tackle the mismatch, as well as a *sample usage*. The template can be used both as guideline for developers and as input to a tool that automatically generates the adapter code.

We distinguish between two types of mismatches: operation-level mismatches and protocol-level mismatches. Operation level mismatches characterize heterogeneities related to operation definitions. Such differences that occur when two services S and SR have operations with the same functionality but differ in operation name, number, order or type of input/output parameters. Protocol level mismatches characterize heterogeneities related to message choreography and temporal/transaction properties. Examples include differences that occur when two services expect a message in different order, when one service sends messages that the other does not accept, when one service requires a single message to achieve certain functionality while the other requires several, and so on. A comprehensive discussion is provided in (Benatallah, Casati, Grigori, Motahari-Nezhad, and Toumani, 2005).

4 Discussion

There has been substantial efforts and progress in the area of Web services, most of which has been focused on service description models and languages, standards, and on automated service discovery and composition (Alonso, et al. 2004). Recently, authors have published papers that discuss similarity and compatibility at different levels of abstractions of service specifications (Bordeaux et al. 2004, Wombacher, et. al 2004, Dong, et. al. 2004). The proposed approaches do not provide fine-grained analysis operators. In addition, they do not consider message and protocol heterogeneities. In terms of protocols specification and analysis, existing approaches provide models (e.g., based on pi-calculus or state machines) and mechanisms (e.g., protocols compatibility and replaceability checking) to compare specifications for software components (Yellin, Strom, 1997, Canal, et. al. 2003). These efforts can be leveraged for Web service protocols, but are not sufficient. In fact, service protocols require richer description models than component interfaces, as clients and services are typically autonomous and therefore service descriptions are all that client developers have to understand to know how the service behaves. Our framework builds upon existing protocol models and techniques to provide high level abstractions and operators for service protocols analysis and management. In addition, a component of the tool enables automated code generation (BPEL skeletons) from protocol models, therefore provides support for model-driven and automated Web service development (Baina, Benatallah, Casati, Toumani, 2004). This framework has been implemented in a prototype platform, called

ServiceMosaic, as a CASE toolset for modelling, analysing, and managing service models including business protocols, orchestrations, and adapters. The ServiceMosaic platform is developed using Java and J2EE technologies in the Eclipse platform (www.eclipse.org).

Our current work focus is on extending analysis and management techniques for timed protocols (Benatallah, Casati, Ponge, and Toumani, 2005), and we will next concentrate on transactional aspects. We are also investigating business protocol discovery techniques to bring the benefits of protocols based interactions to services that do not explicitly model business protocols, or to interactions that involve groups of services. Finally, we plan to explore techniques for cataloging and analyzing previous adapters to improve the process of developing new adapters.

Acknowledgment. The work presented here, which is a part of ServiceMosaic project, is performed in collaboration of Fabio Casati, Farouk Toumani, and Julien Ponge.

5 References

- Papazoglou, M.P. and Georgakopoulos, D. (2003): Special Issue on Service-Oriented Computing, *Communications of ACM*, (46)10, ACM Press.
- Alonso, G., et al. (2004): *Web Services: Concepts, Architectures, and Applications*. Springer Verlag.
- Benatallah, B., Casati, F., Grigori, D., Motahari-Nezhad, H. R., and Toumani, F. (2005): Developing Adapters for Web Services Integration, *Proc. of CAiSE'05*, Springer.
- Benatallah, B., Casati, F., Ponge, J. and Toumani, F. (2005): On Temporal Abstractions of Web Services Protocols, *Proc. of CAiSE Forum*. Springer.
- Benatallah, B., Casati, F., and Toumani, F. (2005): Representing, Analysing and Managing Web Service Protocols, *Data and Knowledge Engineering Journal*.
- Benatallah, B., Casati, F., and Toumani, F. (2004): Web Service Conversation Modeling: A Cornerstone for e-Business Automation, *IEEE Internet Computing*, (8)1. IEEE Press.
- Rahm, E., and Bernstein, P. A. (2001): On Matching Schemas Automatically, *VLDB Journal*, 10 (4).
- Yellin, D.M., and Strom, R.E. (1997): Protocol Specifications and Component Adaptors, *ACM TOPLAS*, 19(2).
- Baina, K., Benatallah, B., Casati, F., and Toumani, F. (2004): Model-Driven Web Service Development, *Proc. of CAiSE'04 Conference*, Springer.
- Canal, L., et. al. (2003): Adding Roles to CORBA Objects, *IEEE Transaction on Software Engineering*, 29(3).
- Bordeaux et al (2004): When are two Web Services Compatible? *VLDB TES Workshop Proceedings*.
- Wombacher, A., et. al (2004): Matchmaking for Business Processes based on Choreographies. *In proc. of EEE*. Taipei, Taiwan.
- Dong, X., et. al. (2004): Similarity Search for Web Services. *VLDB Conference*. Toronto, Canada.

Postmodern Prospects for Conceptual Modelling

James Noble

Computer Science
Victoria University of Wellington
New Zealand
kjx@mcs.vuw.ac.nz

Robert Biddle

Human-Oriented Technology Lab
Carleton University
Canada
robert.biddle@carleton.ca

Abstract

A number of recent developments in software engineering — from agile methods to aspect-oriented programming to design patterns to good enough software — share a number of common attributes. These developments avoid a unifying theme or plan, focus on negotiation between different concerns, and exhibit a high level of context sensitivity. We argue that these developments are evidence of a *postmodern turn* in software engineering. In this paper, we survey a number of these developments and describe their potential implications for the practice of conceptual modelling.

Keywords: Postmodernism, Conceptual Modelling

1 Introduction

A spectre is haunting software development — the spectre of *Postmodernism!* A number of recent developments in software engineering — from agile methods to aspect-oriented programming to design patterns to good enough software — share a number of common attributes. These developments avoid a unifying theme or plan, focus on negotiation between different concerns, and exhibit a high level of context sensitivity.

In this paper, we will argue that these developments are evidence of a *postmodern turn* in software engineering. The notion that software development could usefully be analysed as postmodern was first identified by Hugh Robinson and his colleagues around ten years ago (Robinson, Hall, Hovenden & Rachel 1998). In their prophetic article *Postmodern Software Development* they write:

We have seen that the development methods of software have been grounded in modernism and the Enlightenment, but that this has led to conflicts and contradictions, and a notion of ‘software crisis’ as the modernist metanarrative has broken down.

The views expressed here are that the postmodern ethos can offer some emancipation to the process of software development. In questioning the accepted rules and values and, crucially, not offering any others in their place, software development has to become a more locally negotiated phenomenon. The rules you follow are the ones that are suggested by the situation at hand.

Copyright ©2006, Australian Computer Society, Inc. This paper appeared at Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Australia. Conferences in Research and Practice in Information Technology, Vol. 53. Markus Stumptner, Sven Hartmann and Yasushi Kiyoki, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

Robinson’s article is primarily concerned with the process of software development. More recently, in a series of *Notes on Postmodern Programming* (Noble & Biddle 2002, Biddle, Martin & Noble 2003, Noble & Biddle 2004) we have considered more pragmatic artifacts of software development, programming and programming technology, including languages and design, and the construction of systems from existing software.

In this paper, we draw on this work, plus some crucial theories of the postmodern, to raise questions of postmodernism for conceptual modelling. This paper is in three parts: in the first part, we describe some of the developments and classifications of postmodernism as they have developed in other disciplines, outside software development. Then, in the second part, we will survey a number of developments within software engineering and describe how they exhibit postmodern features. Finally, we conclude by proposing some potential prospects for conceptual modelling faced with the spectre of this postmodern turn. Of necessity, this paper is a survey, rather than an in depth treatment, and to avoid disappointment, be aware that we aim to raise questions, rather than provide answers.

2 Postmodernism

The first question that is generally raised when discussing *postmodernism* is simply: what does postmodernism mean?¹ If modern means “now” or “today”, how can something happen that is *after* what is now or what is today? Indeed, this is a good question: one of the first postmodern theorists, Jean-François Lyotard, describes postmodernism as “*the paradox of the future (post) anterior (modo)*” — that is, the postmodern is what comes after the future (Lyotard 1992, Lyotard 1984, Wikipedia 2005).

While a love of paradox, in-jokes, and irreverence is certainly a typically postmodern style, for the purpose of pedagogy, in this paper we can adopt a more prosaic (but equally unhelpful) definition, derived from the *Notes on Postmodern Programming* (Noble & Biddle 2002, Biddle et al. 2003, Noble & Biddle 2004):

postmodernism applies modern means to other ends

In the case of software development, computer science, software engineering, then, postmodernism often means applying (or misapplying) the fruits of the disciplines’ development over the last fifty years, but applying them a way, or for a purpose, that their original developers would oppose.

¹The traditional answer to this question is that the margin of this paper is insufficient to contain a definition (Noble & Biddle 2002).

In the remainder of this section, we will explore the definition of postmodernism in general; proceeding to considering postmodernisms in software development in the next.

2.1 Incredulity towards Meta-Narratives

In the foreword to his seminal text *The Postmodern Condition*, (Lyotard 1979, Lyotard 1984) the philosopher Jean-François Lyotard characterised postmodernism as “*incredulity towards meta-narratives*”. At little more length, Lyotard writes:

Le recours aux grands récits est exclu; on ne saurait donc recourir ni à la dialectique de l'Esprit ni même à l'émancipation de l'humanité comme validation du discours scientifique postmoderne. Mais, on vient de le voir, le “petit récit” reste la forme par excellence que prend l'invention imaginative, et tout d'abord dans la science.

Recourse to grand narratives is forbidden; we cannot resort to the dialectical of the Spirit nor even to the emancipation of humanity for validation of postmodern scientific discourse. But, as we have seen, the “small narrative” is a form which superbly allows imaginative invention, and most of all in science.

In brief, what Lyotard means is that the sustaining common myths of (Western) civilisation (Christianity, Marxism, Newtonian Physics, Progress) no longer have the strength that they held prior to the twentieth century. There is no longer one *big story* to which all of society may subscribe. To present a brief example, the *Somme* then Auschwitz illustrate that progress, mechanisation, and industrialisation are not themselves necessarily a universal good, the therefore progress is not to be sought as an end in itself. Similarly, the lack of churchgoing in most Western countries, the adoption by most Western Labour parties of private-sector solutions to social problems, and the vicissitudes of relativity (not to mention string theory) illustrate that other grand narratives (Christianity, Socialism, Newtonian Mechanics) are no longer normative. More pragmatically, perhaps, the developments of software technologies and the communication technologies they undergird have resulted in a world where almost everybody is just a phone call away, or virtually present in our living-rooms via television and the Internet. And as recent events continue to show, there is no shared grand narrative by which we may live.

We argue that computer science similarly has had a grand metanarrative: a big story that we are careful to teach first-year students and hold up as the idea of the systems we design and build. In computer science, the primary goals are that software must be correct and efficient — software engineers often add in that the software must be maintainable and usable. The first two introductory concepts of the 2001 ACM Computing Curriculum are *algorithmic computation* (effectively correctness) and *algorithmic efficiency* (The Joint Task Force on Computing Curricula 2001). As Martin Rinard has pithily described, software that fails to meet these goals is seen as evidence of a *moral* weakness on behalf of its programmers (Rinard, Cadar & Nguyen 2005a).

We argue that software development now comes under the postmodern condition: efficiency and correctness, the grand narratives of the discipline, are no longer an effective or useful guide to practice.

2.2 Negotiation and Context Sensitivity

The death of meta-narratives raises a practical problem: without an overarching framework, how can one make decisions? In computer science, if we should not be aiming to build correct software, or efficient software, then what should be doing? Indeed, this is another major critique of postmodernism — that without a metanarrative, all one is left with is uncritical relativism — that “*anything goes*”.

This is a caricature of Lyotard’s position, however. The absence of a single, overriding grand narrative does not mean that decisions may be taken without reference to any external referent. Rather, in place of a single privileged master narrative, we must contend with many localised small narratives, and reconcile them in making a decision. Thus, rather than simply following a grand narrative — Marxist theory, Christian ethics, or in software, a methodology like Structured Design, Stepwise-Refinement, or the Unified Method — we need to determine what small narratives are “in play” at any particular time, and then attempt to resolve the conflict between them. To quote Lyotard (1984):

... the principle that any consensus on the rules defining a game, and the “moves” playable within it must be local, in other words, agreed on by its present players and subject to eventual cancellation.

In practical design or modelling activities, this negotiation between many small narratives typically manifests itself as sensitivity to the context of a model, design, or decision. In architecture, for example, a tenet of postmodern design is sensitivity to the actual location, to the physical site where the building will be constructed. This is in contrast to modern architecture, where (modern) technology dominates all other considerations, and so you build a large rectangular concrete-steel-and-glass box wherever you happen to be (Jencks 1987).

In software design and engineering, for example, postmodern development is more likely to be concerned with existing software products, the values of development and client organisations, the expertise and experience of the development staff, rather than seek to impose an overarching methodology.

2.3 Double Coding

Negotiation between many small stories, and then fitting a design into a particular context (or contexts) leads to one of the key characteristics of postmodern designed *artifacts* — in software in as much as architecture, product design, music, or other art forms. This is that postmodern designs typically use *double coding* (or *multiple coding*) to appeal to more than one audience or meet more than one concern.

The easiest examples to see of this are in architecture: rectangular concrete-and-steel-and-glass boxes are notoriously uncongenial places in which to work and live, not to mention being indistinguishable from each other. Postmodern architects, then, using the same technologies as their modern forebears to actually construct buildings, will add on decorations, spires, domes, arcades, and other ornaments both to distinguish builds from each other, and also to humanise their buildings as homes or workplaces (Jencks 1987).

We find similar multiple codings in software. Contemporary desktop operating systems (Windows, Linux, Macintosh) are good examples here: just as a postmodern architect may paint a mural (or at least

colour the building materials) on the side of a building, add a fancy three-dimensional dome over the doorway, and use contrasting materials for visual effect on walls or floors, so personal computers and mobile phones add fancy screen backgrounds, customised sound effects and ringtones that does not change the functionality of the underlying software but presumably makes it more appealing to its users.

Double coding gives rise to another common criticism of postmodernism — that postmodernism prefers surface to depth, appearance to reality, lying to truth, or cute cross references (Sanrio Company, Ltd. & Sanrio, Inc 2005) to straightforward explanation. And indeed, this criticism is valid to a point (Waugh 1943): one (or more) of the codes are often pastiches of existing or historical styles that are laid over other foundations or applied to other purposes.

Some well-known examples of double-coding in software are the Java programming language and the latest Macintosh operating systems. Java was sold as a successor to C++ — adopting much C++ syntax and terminology. In terms of its underlying technology, Java is effectively a Smalltalk system with types on the bytecodes and an overcomplicated compiler — and indeed, advanced Java programming techniques, relying on garbage collection, finalisation, reflection, dynamic code generation, and so on, are much more closely related to Smalltalk practice than to C++ practice. Similarly, the current (and near future) Macintosh laptops — X86 architecture, open source Darwin kernel, KHTML-based web browser, etc — is almost exactly the same as current Linux or BSD-based systems apart from the details of the design of the window system, and the supremely important logo on the back of every screen.

Double coding and constant negotiation often give rise to a sense of pervasive irony — parody, pastiche, or knowingness is a characteristic of much postmodern writing, architecture, and design. The *Hacker's Dictionary* contains many examples (Raymond & Steele 1993); more recent ones include the pervasive use of cartoons to illustrate computer science textbooks, a programming languages named after a coffee shop, and digital rights management software that rootkits your computer with illegally copied code. This irony is now pervasive: we have operating systems called “Vista”, and positioned with skyscrapers: but we also have operating systems with a fat penguin as a logo, that promote stability as its main advantage.

2.4 Reuse

Finally, where modernism sets itself against past designs and past practice, postmodernism treats the past as just another set of small narratives. Where modern computer science chose to invent a discipline from nothing, a postmodern approach can incorporate all the techniques and experiences from the sixty years the discipline has existed — holding them as contingent, to use as needed.

In some ways, this is evidence of the success of software: we do have a history of successful techniques that have been used to build successful systems. Furthermore, in the last ten to fifteen years, we have also constructed a range of successful software components and subsystems that can be incorporated into new systems. Especially in industrial practice, most development involves existing systems — either for so-called “maintenance” (most often adapting well-functioning successful systems to incorporate new requirements), or as larger-scale developments which graft new functionality and technologies onto existing, long-lived systems, or indeed in some cases replacing systems wholesale.

This has a number of effects in practice. One is that mastery of a programming language, modelling notation, or algorithm or data structure implementations is not enough to equip a professional programmer: rather, programmers and designers also need knowledge of existing systems — sensitivity to the technical context of the development — and of available libraries, components, and frameworks that they can combine into programs.

2.5 Postmodern Responses

In different disciplines and in different contexts, postmodernism responds in different ways to forces described above. Across these disciplines, however, we can generally identify a number of stereotyped responses to postmodernity (or equally different types of postmodernism, or different postmodernisms). The architect and architectural critic Charles Jencks has identified four kinds of architectural postmodernisms. Based upon his classification (Jencks 1987), in this section we outline what we consider the four key software responses to postmodernism: neoclassicism, eclecticism, antimodernism, hypermodernism.

2.5.1 Neoclassicism

Especially within architecture, the most common postmodernism is a variety of *neoclassicism* — also known as postmodern classicism. In postmodern architecture, neoclassicism adopts the standard construction techniques from modern architecture, but adds features drawn from classical architecture to “humanise” the buildings, and to provide a consistent “skin” over the building.

Much contemporary software and language design is neoclassical — simply as a way to smuggle (post) modern technologies into practice. Java or C# are two good examples here: their syntax is carefully designed to look like C or C++, yet their semantics are much closer to Lisp or Smalltalk. This shift is obvious in Apple’s Dylan language, where the first version used Lisp-style S-expression syntax (Shalit 1992), while the second version adopted something much close to C or Pascal (Shalit, Moon & Starbuck 1996).

2.5.2 Eclecticism

The other main postmodern response is *eclecticism* which throws together a number of different features or components, resulting in a collage or melange of textures or styles, with no apparent common thread. IM Pei’s glass pyramid in front of the Louvre is one example: Frank Gehry’s Stata Centre for MIT’s CSAIL lab is another. This response, drawing heavily on quotations or allusions, is common to postmodern music and visual arts, as well as architecture.

A number of postmodern software systems and languages exhibit eclectic postmodernism. The most well-known is the Perl programming language. Perl’s designer, Larry Wall, has explicitly described Perl is the “first postmodern programming language” and discussed how he explicitly and indiscriminately borrowed features from many other programming languages (both “high culture” languages such as Pascal and BCPL, and “low culture” languages such as BASIC or TRAC) to produce Perl’s design. Still within programming languages, two complementary examples are the recent text *Higher-Order Perl* (Dominus 2005) (with a back cover blurb promising to teach Perl programmers “*powerful programming methods — new to most Perl programmers — that were previously the domain of computer scientists*”), and use of Haskell to build DOM Components (Finne, Leijen, Meijer & Jones 1999).

Our previous work, the *Notes* (Biddle et al. 2003), is written in an eclectic form: this paper is neoclassical (it at least *looks* like a scientific paper). The advantage of eclecticism is that — by simply combining different ideas, components, or styles, it doesn't “paper over” significant difficulties or differences between them: a neoclassical approach, as in this paper, can appear to tell a straightforward, consistent story about something that is neither straightforward nor consistent.

2.5.3 Antimodernism

The third postmodern response, which we call *anti-modernism*, is less common in software development, but is more common both in pedagogy and also commercial analysis and modelling practices. This is an essentially low-technology approach, and loosely parallels the “ecological” approach that Jencks identifies in postmodern architecture (Jencks 1987).

An ironic icon for this approach could be Dijkstra, in abandoning the typewriter or word processor and hand-writing all his EWD letters². In pedagogy, techniques such as *Computer Science Unplugged* (Bell, Fellows & Witten 1998) attempt to teach the basics of computer science to schoolchildren or university students without a computer, while Ronald Stamper has advocated *Informatics Without the Computer* as an approach to both systems analysis and an attempt to broaden systems analysis techniques to be applicable to non-computer systems (Stamper 2001). And computer science seems full of curmudgeonly statements of form “X is an improvement over all its successors”.

More practically, low-technology approaches have been advocated as tools, especially for analysis or modelling. Class-Responsibility-Cards (CRC Cards), although originally proposed as a learning tool, are accepted as an effective group-based analysis tool (Beck & Cunningham 1989); Coplien has advocated the use of a whiteboard rather than CASE tools (with a separate “mercenary analyst” role to move information into CASE tools if necessary) (Coplien & Harrison 2004); the design patterns movement has resisted automation or case tool support for patterns (Coplien 1996).

Examples of antimodernism in *systems* design seem to be rather harder to come by — presumably because an antimodern approach would be to eliminate the computer system, and such decisions are not often described in the literature of the discipline. The “Wiki Way” is one successful example of antimodern design (Leuf & Cunningham 2001), allowing users to create web sites by using minimal textual markup within any web browser, rather than complex integrated authoring environments.

2.5.4 Hypermodernism

The last response we will consider is the continuation, or the exaggeration of modern design and aesthetics. One way to think about this is that eclectic postmodernism can coöpt any past or present style: one of those it can coöpt is modernism. Then, in spite of what we argue is the general failure of modern master-narratives of software development, research and development can still continue working directly from modern premises.

The hypermodern response most practically manifests itself as a kind of minimalism: building systems based on a single language or single architecture, which may be correct, very powerful, and very

²Luca Cardelli provides a ironic, eclectic overlay on this refusal, with a “handwritten” postscript printer font called “Dijkstra” (Cardelli 2005)

efficient, but because of a modernist rejection of existing practice, are unable to be directly adopted or used widely. The extensible, customizable self-documenting Lisp-based emacs editor is one such system (Stallman 1981); in spite of all its power and elegance of design, it has never been widely adopted outside a circle of cognoscenti.

The key difference between this extreme postmodern response, and the modern grand narrative, is that, say, while Wirth was developing Pascal could do so with the idea that an independent, completely-Pascal-only system could well become widely used — as indeed it did. Twenty years later, few people were willing to install a new operating system, learn a novel user interface paradigm, and then a new programming language to use Oberon.

3 The Postmodern Turn

Having toured postmodernism, in this section we will turn to software, and consider how some recent developments may be infected with postmodernism. This selection is not intended to be complete or consistent. We have certainly omitted many important developments (Open Source (Raymond 2001), UML, XML, various kinds of multiparadigm programming, customisable methodologies), some of which are discussed elsewhere in the *Notes*.

3.1 Agile Methods

The first postmodern development we consider here is the introduction of so-called Agile software development methods. The most well-known (and arguably most adopted) Agile methodology is Extreme Programming (Beck 1999). The Agile software movement has the advantage of a manifesto, show in Figure 1 (The Agile Alliance 2001):

In postmodern terms, traditional software development projects stands under a grand narrative of “progress”. Expert analysts “extract” or “mine” requirements from clients in roughly same spirit oil and coal are extracted or mined; once the experts have sufficient knowledge of the terrain, they can the proceed to manufacturing the software. Often this manufacturing is carried out in roughly the same techniques as industrial production lines and the end of the process, the final software is shipped to clients and the process is over (Robinson et al. 1998).

Strange as it may seem, the idea of an Agile systems development is that it does not make any progress. Agile development treats all development as maintenance programming, keeping an existing system running, and evolving it through a series of iterations, not as a greenfield development from nothing. Ideally, at the end of any Agile iteration, a system will suit its operational context (functional requirements) as well as its development context (principally resources) allow: the system does not grow more complete or more complex — or even necessarily larger over time (especially as features and their supporting code may be removed once they are no longer required).

In practice, of course, Agile developments do have a model of progress — the number of user stories or functions completed, the amount of “backlog” in a Lean or Scrum development (Poppendieck & Poppendieck 2003, Schwaber & Beedle 2001). But this progress is temporal, limited, and local: as a whole, the system and its developers exist in a steady, *stable* state (Beck 1999).

Thus the key point of the agile manifesto — rather than applying a manufacturing methodology, customers and developers are seen as bringing their own

Manifesto for Agile Software Development

We are uncovering better ways of developing software
by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Figure 1: Agile Manifesto

knowledge about their own spheres of expertise, (their own little narratives) and they continually negotiate changes to the extant working software to be completed in the next iteration. Even though traditional methodologies may build software iteratively, the overall scope and cost of the construction effort is generally estimated and then established in advance. In contrast, Agile developments prefer variable scope contracts, perhaps with a fixed cost-per-unit-time, during which the software will be maintained or evolved. To maintain this focus on the actual software and personal interaction with clients, many Agile methodologies, in particular XP, downplay or actively forbid any attempts to maintain software models, plans, or analysis or design documents. Perhaps whiteboards and notepaper may be permitted, but they must be erased or disposed of at the end of each working day. Ownership of domain knowledge remains with the customer representatives; the structure of the software is only represented by its source code.

In terms of our classification of postmodernisms above, Agile methods at a large scale are antimodernism — eschewing complex modern estimation and planning methods, analysis and design tools and methodologies in favour of direct communication and (ephemeral) paper based aids similar to CRC Cards. At small scales, Agile methods promote adoption of advanced code analysis and programming tools, such as refactoring browsers, that can automatically analyse and reshape code to incorporate new requirements, and unit testing regression to ensure these changes preserve the code's behaviour: in this sense, they are neoclassical. On the other hand, some Agile methods (again, particularly XP) try to promote their own (hypermodernist) grand narratives, advocating strict adherence to all the practices to be considered “truly” practicing XP (Portland Patterns Repository 2005). In our research into actual Agile development teams (Martin, Biddle & Noble 2004a, Martin, Biddle & Noble 2004b), however, we have found that most development teams in practice take a rather more flexible approach, adopting those practices that are perceived to fit within their development context, and ignoring those that do not.

3.2 Aspect-Oriented: Software Development without Knowledge

Traditional software development effectively enjoys a unified ontology: software has single hierarchical program structure, a tree (Dahl & Hoare 1972). Depending on the type of program, this may be based either on a functional decomposition (as in structured programs); part-whole aggregation hierarchies (common in entity-relationship modelling and non-

relational databases) or classification hierarchies (especially object-oriented modelling). We consider that this structure represents a grand narrative: a program is a single unitary artifact, a consistent encoding of domain knowledge gained by a modeller and created by a programmer.

Aspect-Oriented Software Development (AOSD, also known as Advanced Separation of Concerns) is based upon a fundamental critique of this structure for programs (Kiczales, Lamping, Mendhekar, Maeda, Lopes, Loingtier & Irwin. 1997, Harrison & Ossher 1993). AOSD points out that programs in fact consist of multiple interlocking issues or concerns, so that each individual single program element in a traditional tree structure (say a class definition) will *tangle* code related to a number of different concerns (data storage, database access, user interfaces) while any single concern will be *distributed* across many different program elements. AOSD claims this tangling and distribution causes a range of problems with program comprehension, maintenance, and construction, and as a result, proposes that program structures should be multifaced and multilinked, rather than tree shaped. Programming language constructs need to be revised to support these new structure, so that programs can break down programs' structure into many small components, each corresponding to an individual concern, and then somehow combining those parts back together into a functioning program.

In fact, AOSD's critique is not new, harking back to Fred Brooks' description of the “*intricately interlocking software elephant*” (Brooks 1987). Neither is its solution new: the phrase: “*separation of concerns*” has a long history within computer science (Dijkstra 1982, Parnas & Clements 1986). The difference in AOSD is that this decomposition is heterarchical, rather than hierarchical: a graph, rather than a tree³. Each separate component or separate concern plays the role of a small narrative in the program's design: modelling or programming languages explicitly have to embody strategies to negotiate between this local narratives (such as composition rules in subject-oriented programming, or aspect dominance in Aspect/J).

This kind of cross-cutting, interlinked structure is common to many different postmodern critiques, including antimodernism architect Christopher Alexander's description that “*A city is not a tree*” (Alexander 1965), or eclectic postmodern philosophers Deleuze and Guattari's notion of a rhizome (Deleuze & Guattari 1987):

³Observant readers will realise that the *implicit* structure of any program is a (multi)graph, and modern, structured languages restrict only the *explicit* structure of a program to be a tree. AOSD proponents argue this is precisely the point: the aim of AOSD is to make explicit as much of the structure of the program as possible.

The rhizome itself assumes very diverse forms, from ramified surface extension in all directions to concretion into bulbs and tubers. When rats swarm over each other. The rhizome includes the best and the worst: potato and couchgrass, or the weed. Animal and plant, couchgrass is crabgrass.

AOSD programs in current practice can be considered neoclassical, as typically they consists of “base” programs in a modern programming language with a tree topology, but with extra aspects being weaved in to handle one or to extraordinary functions, such as program tracing, monitoring, debugging, concurrent synchronisation, or security. More recent developments ranging from formal models (Aldrich 2005) to modelling all interobject relationships as aspects (Pearce & Noble 2006), lean more towards hypermodernism, capturing as much of a programs’ implicitly interwoven structure explicitly in aspects.

3.3 Design Patterns

Existing methodologies are also based on grand narratives of design, describing how to construct entire programs by developing requirements top down into a design then into consistent code. This is as true for traditional structured methodologies (Yourdon & Constantine 1979), object-oriented methodologies (Wirfs-Brock, Wilkerson & Wiener 1990), or even aspect-oriented methodologies (Clarke & Baniassad 2005). In that sense, methodologies embody an ethic of what is “good” over whole programs, and so design all of a program or system in the same way.

In contrast, design patterns (Gamma, Helm, Johnson & Vlissides 1994) are little local rules about what is good in a design, or rather, that describe and evaluate a characteristic, reusable design in such a way that programmers can apply this design to their programs when it is needed. The classic definition of a pattern, “A solution to a problem in a context” both encapsulates the specificity of a pattern (one solution to one problem) and also its context dependence (Coplien 1996, Gamma et al. 1994). Because patterns are small, specific, and contingent, they can be adopted by programmers without changing their existing methodologies or work methods, as and when required, irrespective of what other design discipline has been adopted. Patterns may be employed singularly, or in combination with either each other, or with other techniques or programming constructs or techniques. As John Vlissides describes in his *Pattern Hatching* (Vlissides 1998):

You will discover techniques for applying certain patterns and not applying others, as circumstances dictate.

In terms of the typology of postmodern responses above, design patterns are generally an eclectic postmodernism, as multiple patterns from multiple sources may be used indiscriminately together in a program — indeed, we consider this is a key reason for the patterns movement’s success. Parts of the patterns movement are antimodernist, in that, like Agile development, they eschew automated support (or indeed any kind of external analysis) of patterns (Coplien 1996). Refactoring, which emerged from the same milieu as patterns, is not antimodernist in this way. On the other hand, also like some XP and AOP proponents, those patterns proponents who follow Christopher Alexander (the originator of design patterns in Architecture (Alexander 1979, Alexander 1977)) are simultaneously hypermodernist and antimodernist, on one hand, insisting that all patterns

must be joined together in a grand, overarching *pattern language* — another grand narrative — and on the other, dismissing academic analysis or automated support for program design

Patterns are also related to a number of other programming techniques that have similar effects. Refactorings (Opdyke & Johnson 1990, Fowler 1999) are generally smaller than patterns, describing localised code changes that improve (or at least modify) the structure of a program. Analysis Patterns (Fowler 1997) and Problem Frames (Jackson 2001) are again descriptions of particular localised, partial solutions but addressed earlier in the traditional software lifecycle.

3.4 Good Enough Software

We discussed above that key parts of the Computer Science grand narrative are the twin goals of correctness and then efficiency: these are non-negotiable — although correctness is more important. From this perspective, the aim of developing only “good enough software” — software that is neither correct nor efficient — but is good enough for its context of use, is also a postmodern approach.

Gabriel’s essay *Lisp: Good News, Bad News, How to Win Big* (Gabriel 1991) includes an early elucidation of this argument, arguing that a design that emphasised simplicity and sacrificed completeness would have better survival characteristics than a design that promoted correctness and completeness.

Agile approaches, such as XP, take this reasoning further: accepting that all design qualities — correctness, efficiency, completeness, consistency, simplicity, development time, and cost — are independent variables in a development process (Beck 1999). A particular development project needs to negotiate between all these small narratives of software characteristics to fit the projects’ context, perhaps trading off quality for price, for development time, for the project scope (for a simple website development) or perhaps delivering correctness at any cost (for a life-critical system). Rinard’s Acceptability-Oriented Computing (Rinard, Cadar & Nguyen 2005b) goes one step further, arguing that the programming and design techniques programmers typically adopt to ensure programs’ correctness actually reduce the correctness and reliability of the systems of which those programs are parts.

The concept of good enough software is an antimodernist response to development pressure: rejecting the modern claims that software should be correct and efficient. The complementary contrary aim of developing perfect software through the use of formal methods is the hypermodern antithesis to this. Perhaps there is also a neoclassical synthesis, where tools like Findbugs (Hovemeyer & Pugh 2004) or Spec# (Barnett, Leino & Schulte 2004) employ sophisticated formal analyses to catch potential errors, but do not guarantee the correctness of the resulting program.

3.5 Scrap-Heap Programming

Steven Conner (Connor 2004) describes postmodernism as

...that condition in which, for the first time, and as a result of technologies which allow the large-scale storage, access, and reproduction of records of the past, the past appears to be included in the present.

Nowhere is this aspect of postmodernism more explicit than in software development. This is for two reasons: first, digital technologies are the key enablers of this large-scale storage, access, and reproduction;

and second, that software itself is the content *par excellence* that can be stored and exchanged using these technologies. In effect, everything is becoming software.

This shift has a major impact on software development. The context of a software development projects now includes huge amounts of existing software, available via Google on the the web, provided by the open source movement, purchased one way or another from software vendors — and, often most pertinent — pre-existing software within the client organisation which the new software must extend, cooperate, or replace. To self-plagiarise: “*How do you write a program when every program has already been written*” (Noble & Biddle 2004).

Our techniques for modelling and programming must take account of this existing software — and in particular, of the changes it makes to the structure of the software that we build. To return again to our theme: once software is constructed mostly out of other software, there is no longer any one grand narrative, big story, of software structure: no longer any one coherent software design. Rather, we build software in a scrapheap (Noble & Biddle 2002, Moore & Pryce 2005, Brucker-Cohen & Moriwaki 2005, Channel 4 2004). The end result is the “*Programmer as DJ*” — (re)mixing existing systems, scavenging parts of systems, buying or building discrete components and then writing or generating amorphous glue that binds these disparate parts together.

The modern grand narratives of program design generally work from requirements to design to engineering. This model holds good for both Agile and non-Agile methodologies: in fact, Agile methodologies such as XP often have an even stronger focus on this separation of labour: a customer provides requirements while the programmer implements them (Beck 1999). Much postmodern software engineering tends to work bottom up — not bottom up from the requirements, but bottom up from the scrapheap. Programmers begin with the components that they have to hand, or can find on Google, or can scavenge; then work out what they could build out of the components; and only then negotiate with customers over requirements.

Actual assembly of scrapheap software is often not via programming in traditional programming languages, but rather via scripting languages, languages that are good enough for the job. Traditional programming languages are designed for writing whole programs: scripting languages are designed for writing small parts of them: the interstitial filler between the parts Google delivers from the scrapheap.

These little languages are low culture languages — shell scripts, Perl, Ruby, Visual Basic — generally ignored by academic computer scientists, software engineers, or modellers. The languages are interpreted and dynamically typed, good enough in that they are concerned neither with correctness (offered by static typing) or efficiency (compilation) — but are nevertheless powerful and efficient enough to write full applications, given that they make it very easy to connect together other software. Where these languages are discussed (Bently 1988), they are generally not studied in their own right, but rather as implementation techniques to build a new customised language for a particular problem. While building a new language is a scrapheap technique, the most common case is to simply use one or more of these languages to glue existing components together.

Brian Foote has captured the Scrapheap dynamic as the difference between a *Big Ball of Mud* (Foote & Yoder 2000) and a *Big Bucket of Glue* (Foote 2005). Foote describes LISP as a Big Ball of Mud — a very flexible syntax, language, system, to which program-

mers can always add more Lisp code because it always remains just a Big Ball of Mud. But the important thing about a Big Bucket of Glue is not the glue: the glue is what sticks together the stuff pulled from the scrapheap. Two buckets of glue can be very different, depending upon what scraps they contain. Programs are composites of multiple things, a conglomerate rather than a mixture, different parts, stuck together with glue that is different again from any of the components: mud is undifferentiated.

So we have composed software, assembled software, software bricolage, (Biddle et al. 2003) — software made out of a range of parts of different sizes, languages, platforms, and technologies. These parts may not necessarily have been designed to be encapsulated reusable components: they will have (implicit) dependencies on architecture, standards, other software that may only be partially understood. Building programs out of existing software results in radically heterogenous systems — because the parts are not just simple “components” (Cox 1990) but may be whole other applications, web services, entire complex legacy interlinked computer systems accessed via screen-scrapers, facades, or wrappers of various sorts. The cost of merely understanding these software, let alone reengineering them to fit some overarching software architecture, would be prohibitive.

Scrapheap programming is in some ways the most obviously characteristic postmodern programming style. In terms of the postmodernisms above, Scrapheap programming is eclectic, highly context sensitive, mixes high and low culture (using anything that will do the job, from Haskell to Visual Basic) and heavily double coded (facades, wrappers, user interfaces are ubiquitous to make scrapheap software seem like something it is not).

4 Prospects for Postmodern Conceptual Modelling

The rubric for the First Asia-Pacific Conference on Conceptual Modelling (Hartmann & Roddick 2004) states:

The actual amount, complexity and diversity of information are increasing day by day. Information that has to be conceptualised and efficiently organised in order to be useful. Conceptual modelling is fundamental to the development of up-to-date information and knowledge-based systems.

What, then, are the prospects for a postmodern approach to conceptual modelling? First, it seems that this rubric already accepts one of the tenets of postmodernism — the increasing volume, complexity, and diversity of information, and perhaps an increasing anxiety to keep our systems always “up-to-date” with a postmodernity that is always just after the future we imagined when we first designed the systems. Second, though, the statement stands on a modern assertion of metanarrative: information has to be *conceptualised and efficiently organised*, in order to be useful. Arguably, Google provides a counterexample: information can be useful given only moderately efficient access, without correctness (conceptualisation) or efficient organisation, and certainly without a grand narrative to hold everything together.

Although, in postmodernity, there is likely no privileged “high ground” that conceptual modelling can claim, this does not mean it has no place in postmodern software development. Rather, conceptual modelling can negotiate as a little narrative, along with many other techniques for the “*development of up-to-date information and knowledge-based systems*” —

with Visual Basic and category theory and Microsoft Bob and abstract interpretation and CRC cards and Haskell.

But, how could such negotiation work in practice? How do you practice conceptual modelling for Visual Basic programs, or for an Agile methodology that explicitly rejects modelling and design? Thankfully, the prospects for aspect-oriented conceptual modelling seem a little brighter, given conceptual modelling's existing move from its straightforward entity-relationship heritage to encompass some object-oriented techniques, and more recently towards XML. How can conceptual modelling adapt to projects where other narratives (cost or time to market) outweigh the traditional values of correctness and efficiency? Ultimately, what utility is there in conceptual modelling a system built piecemeal out of junk found from a scrapheap via Google?

Perhaps it is more useful, as we have done to some extent here, to consider how a particular technology or technique responds to postmodernity, rather than simply asking whether or not something is "postmodern" (especially where "postmodern" is often meant as a synonym for "bad"). If our analysis is correct, then we are all working in postmodernity: an anti-modern response — clinging heroically, desperately, to one ideal imagined future where all information is conceptualised and all databases are normalised — is as much a postmodern response as gleefully claiming eclecticism removes the need (or responsibility) for developers or modellers to make any value decisions. Rather, the point is that every decision is now a local negotiation between various contingent narratives. So, as researchers, we can ask "how is this postmodern?", or "what aspects of this are postmodern?", or "what kind of postmodern response is this?". How is conceptual modelling sensitive to context? What kind of double or multiple codings are in effect? How can conceptual modelling, aware of its status as one little narrative amongst many, contribute to the ongoing negotiation at the core of systems development in the postmodern era?

References

- Aldrich, J. (2005), Open modules: Modular reasoning about advice, in 'ECOOP Proceedings'.
- Alexander, C. (1965), 'A city is not a tree', *DESIGN* pp. 46–55.
- Alexander, C. (1977), *A Pattern Language*, Oxford University Press.
- Alexander, C. (1979), *The Timeless Way of Building*, Oxford University Press.
- Barnett, M., Leino, K. R. M. & Schulte, W. (2004), The Spec \sharp programming system: An overview, in G. Barthe, L. Burdy, M. Huisman, J.-L. Lanet & T. Muntean, eds, 'Construction and Analysis of Safe, Secure, and Interoperable Smart Devices (CASSIS)', Vol. 3362 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 49–69.
- Beck, K. (1999), *Extreme Programming Explained: Embrace Change*, Addison-Wesley.
- Beck, K. & Cunningham, W. (1989), A laboratory for teaching object oriented thinking, in 'OOPSLA Proceedings'.
- Bell, T., Fellows, M. & Witten, I. (1998), *Computer Science Unplugged: The Original Activities Book*, Computer Science Unplugged, Morrisville, NC. www.unplugged.canterbury.ac.nz.
- Bently, J., ed. (1988), *More Programming Pearls*, Addison-Wesley.
- Biddle, R., Martin, A. & Noble, J. (2003), 'No name: Just notes on software reuse', *SigPlan Notices: Proceedings of the Oopsla Onward Track* **38**(2), 76–96.
- Brooks, Jr., F. P. (1987), 'No silver bullet: Essence and accidents of software engineering', *IEEE Computer* **20**(4).
- Brucker-Cohen, J. & Moriwaki, K. (2005), 'Scrapyard challenge workshop', <http://www.-scrapyardchallenge.com/>.
- Cardelli, L. (2005), 'Dijkstra postscript font', <http://www.luca.demon.co.uk/Fonts.htm>.
- Channel 4 (2004), 'Scrapheap challenge', <http://www.channel4.com/science/microsites/S/-scrapheap/>.
- Clarke, S. & Baniassad, E. (2005), *Aspect-Oriented Analysis and Design: The Theme Approach*, Addison-Wesley.
- Connor, S., ed. (2004), *The Cambridge Companion to Postmodernism*, Cambridge University Press.
- Coplien, J. O. (1996), *Software Patterns*, SIGS Management Briefings, SIGS Press.
- Coplien, J. O. & Harrison, N. B. (2004), *Organizational Patterns of Agile Software Development*, Prentice Hall PTR.
- Cox, B. J. (1990), 'Planning the software industrial revolution', *IEEE Software* .
- Dahl, O.-J. & Hoare, C. A. R. (1972), Hierarchical program structures, in O.-J. Dahl, E. W. Dijkstra & C. A. R. Hoare, eds, 'Structured Programming', Academic Press.
- Deleuze, G. & Guattari, F. (1987), *A Thousand Plateaus: Capitalism and Schizophrenia*, University of Minnesota Press. Translated from the French by Brian Massumi.
- Dijkstra, E. W. (1982), On the role of scientific thought, in 'Selected Writings on Computing: A Personal Perspective', Springer-Verlag, pp. 60–66.
- Dominus, M. J. (2005), *Higher-Order Perl*, Elsevier Science & Technology Books.
- Finne, S., Leijen, D., Meijer, E. & Jones, S. P. (1999), Calling hell from heaven and heaven from hell, in 'Proceedings of the Fourth ACM SIGPLAN International Conference on Functional Programming (ICFP-99)', ACM, pp. 114–125.
- Foote, B. (2005), Big bucket of glue (breakthrough idea), in 'OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications', ACM Press, pp. 76–86.
- Foote, B. & Yoder, J. (2000), Big ball of mud, in N. Harrison, B. Foote & H. Rohnert, eds, 'Pattern Languages of Program Design', Vol. 4, Addison-Wesley, chapter 29, pp. 653–692.
- Fowler, M. (1997), *Analysis Patterns*, Addison-Wesley.
- Fowler, M. (1999), *Refactoring: Improving the Design of Existing Code*, Addison-Wesley.

- Gabriel, R. P. (1991), 'LISP: Good news, bad news, how to win big', *AI Expert* **6**(6), 30–39.
- Gamma, E., Helm, R., Johnson, R. E. & Vlissides, J. (1994), *Design Patterns*, Addison-Wesley.
- Harrison, W. & Ossher, H. (1993), Subject-oriented programming (a critique of pure objects), in 'OOPSLA Proceedings', pp. 411–428.
- Hartmann, S. & Roddick, J. (2004), 'The first asia-pacific conference on conceptual modelling (call for papers)', <http://apccm.massey.ac.nz/apccm04/>.
- Hovemeyer, D. & Pugh, W. (2004), Finding bugs is easy, in 'OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications', ACM, pp. 132–136.
- Jackson, M. (2001), *Problem Frames: Analyzing and Structuring Software Development Problems*, Addison-Wesley.
- Jencks, C. (1987), *The Language of Post-Modern Architecture*, Academy Editions.
- Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M. & Irwin, J. (1997), Aspect oriented programming, in 'ECOOP Proceedings'.
- Leuf, B. & Cunningham, W. (2001), *The Wiki Way*, Addison-Wesley Publication Co. <http://wiki.org/>
- Lyotard, J.-F. (1979), *La Condition postmoderne: Rapport sur le savoir*, Collection: "Critique.", Minuit, Paris.
- Lyotard, J.-F. (1984), *The Postmodern Condition: A Report on Knowledge*, Vol. 10 of *Theory and History of Literature*, University of Minnesota Press. Translated from the French by Geoff Bennington and Brian Massumi.
- Lyotard, J.-F. (1992), From the postmodern condition, in A. Easthope & K. McGowan, eds, 'A Critical And Cultural Reader', Allen & Unwin.
- Martin, A., Biddle, R. & Noble, J. (2004a), When XP met outsourcing, in J. Eckstein & H. Baumeister, eds, 'Proceedings of the Fifth International Conference on eXtreme Programming and Agile Processes in Software Engineering'.
- Martin, A., Biddle, R. & Noble, J. (2004b), The XP customer role in practice: Three case studies, in 'Proceedings of the Second Agile Development Conference'.
- Moore, I. & Pryce, N. (2005), 'Scrapheap challenge: A workshop in post-modern programming', Workshop at OOPSLA 2005, San Diego. <http://postmodernprogramming.org/>.
- Noble, J. & Biddle, R. (2002), Notes on postmodern programming, in R. Gabriel, ed., 'Proceedings of the Onward Track at Oopsla 02, the ACM conference on Object-Oriented Programming, Systems, Languages and Applications', <http://www.dreamsongs.org/>, Seattle, USA, pp. 49–71.
- Noble, J. & Biddle, R. (2004), Notes on notes on postmodern programming: radio edit., in 'OOPSLA Companion, proceedings of the Onward! stream', pp. 112–115.
- Opdyke, W. F. & Johnson, R. J. (1990), Refactoring: An Aid in Designing Application Frameworks, in 'Symposium on Object-Oriented Programming Emphasizing Practical Applications', ACM-SIGPLAN, pp. 145–160.
- Parnas, D. L. & Clements, P. C. (1986), 'A rational design process: How and why to fake it', *IEEE Transactions on Software Engineering* **12**(2), 251–257.
- Pearce, D. & Noble, J. (2006), Relationship aspects, in 'Aspect-Oriented Software Development (AOSD)'.
- Poppendieck, M. & Poppendieck, T. (2003), *Lean Software Development: An Agile Toolkit for Software Development Managers*, Addison-Wesley.
- Portland Patterns Repository (2005), 'You Aren't Extreme — Portland Patterns Repository', <http://c2.com/cgi/wiki?YouArentExtreme>. [Accessed November 2005].
- Raymond, E. S. (2001), *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly & Associates.
- Raymond, E. & Steele, G. L. (1993), *The New Hacker's Dictionary*, second edn, MIT Press.
- Rinard, M., Cadar, C. & Nguyen, H. H. (2005a), 'Exploring the acceptability envelope', Presentation to the OOPSLA 2005 Onward! Stream.
- Rinard, M., Cadar, C. & Nguyen, H. H. (2005b), Exploring the acceptability envelope, in 'OOPSLA Companion, proceedings of the Onward! stream'.
- Robinson, H., Hall, P., Hovenden, F. & Rachel, J. (1998), 'Postmodern software development', *The Computer Journal* **31**, 363–375.
- Sanrio Company, Ltd. & Sanrio, Inc (2005), 'The official Sanrio website. home of Hello Kitty', <http://www.sanrio.com/>.
- Schwaber, K. & Beedle, M. (2001), *Agile Software Development with SCRUM*, Prentice-Hall.
- Shalit, A. (1992), *Dylan: an object oriented dynamic language*, first edn, Apple Computer, Inc.
- Shalit, A., Moon, D. & Starbuck, O. (1996), *The Dylan Reference Manual: The Definitive Guide to the New Object-Oriented Dynamic Language*, first edn, Addison-Wesley.
- Stallman, R. M. (1981), EMACS the extensible, customizable self-documenting display editor, in 'Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation', pp. 147–156.
- Stamper, R. (2001), Organisational semiotics: Informatics without the computer?, in K. Liu, R. Clarke, P. B. Andersen & R. Stamper, eds, 'Information, organisation and technology: Studies in organisational semiotics', Kluwer Academic Publishers, pp. 115–171.
- The Agile Alliance (2001), 'The Agile manifesto', <http://agilemanifesto.org/>.
- The Joint Task Force on Computing Curricula (2001), 'Computing curriculum 2001', *Journal on Education Resources in Computing* **1**(3es), 1.

- Vlissides, J., ed. (1998), *Pattern Hatching: Design Patterns Applied*, Addison-Wesley.
- Waugh, E. (1943), *Scoop*, Penguin Books.
- Wikipedia (2005), 'Postmodernism — wikipedia, the free encyclopedia', <http://en.wikipedia.org/>. [Accessed November 2005].
- Wirfs-Brock, R., Wilkerson, B. & Wiener, L. (1990), *Designing Object-Oriented Software*, Prentice-Hall.
- Yourdon, E. & Constantine, L. L. (1979), *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, facsimile edn, Prentice Hall.

Network Data Mining: Methods and Techniques for Discovering Deep Linkage between Attributes

John Galloway^{1,2} and Simeon J. Simoff^{3,4}

¹Complex Systems Research Centre, University of Technology Sydney
PO Box 123 Broadway NSW 2007 Australia

john.galloway@uts.edu.au

²Chief Scientist, NetMap Analytics Pty Ltd,
52 Atchison Street, St Leonards NSW 2065 Australia

³Faculty of Information Technology, University of Technology Sydney

PO Box 123 Broadway NSW 2007 Australia

simeon@it.uts.edu.au

⁴Electronic Markets Group, Institute for Information and Communication Technologies, University of Technology Sydney - PO
Box 123 Broadway NSW 2007 Australia

<http://research.it.uts.edu.au/emarkets>

Abstract. Network Data Mining identifies emergent networks between myriads of individual data items and utilises special algorithms that aid visualisation of ‘emergent’ patterns and trends in the linkage. It complements conventional data mining methods, which assume the independence between the attributes and the independence between the values of these attributes. These techniques typically flag, alert or alarm instances or events that could represent anomalous behaviour or irregularities because of a match with pre-defined patterns or rules. They serve as ‘exception detection’ methods where the rules or definitions of what might constitute an exception are able to be known and specified ahead of time. Many problems are suited to this approach. Many problems however, especially those of a more complex nature, are not well suited. The rules or definitions simply cannot be specified. For example, in the analysis of transaction data there are no known suspicious transactions. This chapter presents a human-centred network data mining methodology that addresses the issues of depicting implicit relationships between data attributes and/or specific values of these attributes. A case study from the area of security illustrates the application of the methodology and corresponding data mining techniques. The chapter argues that for many problems, a ‘discovery’ phase in the investigative process based on visualisation and human cognition is a logical precedent to, and complement of, more automated ‘exception detection’ phases.

Introduction

The proliferation of data is both an opportunity and a challenge. It provides the details that businesses need to solve problems and gain market advantage, that organisations need to improve their operations, that banks and financial institutions need to fight fraud and that governments need to uncover criminal and terrorists activities. At the same time, a large volume of data with different storage systems, multiple formats and all manner of internal complexity can often hide more than it reveals. Data mining – “the process of secondary analysis of large databases aimed at finding unsuspected relationships that are of interest or value to the database owners” (Klösigen and Zytkow 2002) (p. 637) – emerged as an “eclectic discipline” (Klösigen and Zytkow 2002) that addresses these large volumes of data. Earlier data mining technologies have been primarily focused on the analysis of structured data (Fayyad, Piatetsky-Shapiro and Smyth 1996; Han and Kamber 2001; Hand, Mannila and Smyth 2001). Although the data mining researchers have developed methods and techniques that support a variety of tasks, the main interest of analytics practitioners has been focused on predictive modelling. The dominant scenario in predictive modelling is the “black box” approach, where we have a collection of inputs and one or more outputs, and we try to build an algorithmic model that estimates the value of the outputs as a function of the values of the inputs. There are several measures of model quality (Weiss and Zhang 2003), with the accuracy of predictions remaining as a key measure of model quality, rather than the theory that may explain the phenomena.

Fig. 1 illustrates the concepts in terms of a simple example of data about a group of college friends. The data table includes the following columns: name of the student; colour of hair; height and weight; a record of whether the student has been using lotion when exposed to sun; a record of whether the student gets sunburned when on the beach; a record about the proximity of the living locations of the students; transaction reference numbers; and student address. The double dotted line contours the portion of the data table which will be considered in the predictive modelling approach. As the “Name” column contains unique identifiers, it will be ignored, and the data mining task will be to develop a model of the student from this college with respect to the attributes “hair”, “height”, “weight”, “lotion” and “on the beach”. In an unsupervised approach, the students will be clustered into groups and the analyst ends up with the description of the different groups. In this case, the analyst is interested in predicting whether a new student will get sunburned when visiting the beach. The attribute “on the beach” is selected as the “output” (or “target”) and the

attributes “hair” to “lotion” form the input vector. Given the values for the attributes “hair” to “lotion” for a new student, the resultant classifier should be able to predict whether the student will get sunburned or not. The key measure of the quality of the model is the accuracy of predictions, rather than the theory that may explain the phenomena through the relations between the values of the attributes.

In practice, the focus on predictive accuracy in the “black box” approach (inherited from regression analysis and automatic control systems) makes perfect sense. For example, the more accurate a classifier of tumours is based on the characteristics of mammogram data, the better the aid it can provide to young practitioners (Antonie, Zaiane and Coman 2003). In business areas like marketing, such an approach makes sense (for example, determining which few books Amazon should also offer to someone who searches for a particular book). Numerous examples from different areas of human endeavour, including science and engineering, are presented in (Nong 2003).

The theory explaining the “black box”, i.e. how and why inputs are related to outputs, is often of secondary importance to predictive accuracy. However, data mining applications in many areas, including businesses and security (Nong 2003) require deeper understanding of the phenomena. Such complex phenomena can be modeled with techniques from the area of complex systems analysis.

The network perspective with respect to a data set is illustrated in Fig. 1. The structural component of the data set describes explicitly some relationships between the individual entities in the data (in our example in Fig. 1, the column “Lives near” *explicitly* represents the relation of physical proximity between the areas where each student lives). Social network analysis (Wasserman and Faust 1994; Scott 2000) deals with this type of data analysis.

The “classical” data mining table

“input”					“output”				
Name	Hair	Height	Weight	Lotion	On the beach	Lives near	Transaction ref #	Address	
Carmen	blonde	average	light	no	sunburned	Ricard, Karla	14065	24 Jones Av	
Joan	blonde	tall	average	yes	none	Francesca	28501	163 Austin St	
Francesca	brown	short	average	yes	none	Joan	18345	24 Jones Av	
Jose	blonde	short	average	no	sunburned	Ricard, Karla	14065	285 Park Rd	
Jose	blonde	short	average	no	sunburned	Ricard, Karla	28501	285 Park Rd	
Ricard	ginger	average	heavy	no	sunburned	Carmen, Jose, Karla	26912	103 Potts St	
Karla	brown	tall	heavy	no	none	Carmen, Jose, Ricard			
Isabel	brown	average	heavy	no	none	Manu	11995	103 Potts St	
Manu	blonde	short	light	yes	none	Isabel	26912	24 Jones Av	

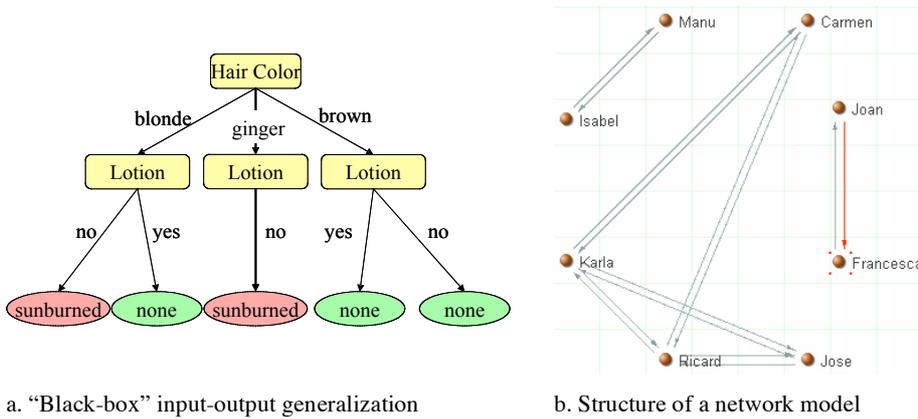
The structural component for social network analysis
The network data mining components

Fig. 1. Different views at a data collection: the “classical” data mining view, the social network analysis and the network data mining view.

In addition to the explicitly coded relationships, there often are *implicit* relationships between the entities described by the data set, especially in the realm of transactions data. Any attribute can come into play for establishing such relations depending on the point of investigation. In our example in Fig. 1, the two attributes “Transaction reference #” and “Address” have been used to look for possible links between the college students. The revealing of such implicit relationships between entities and the discovery of ‘buried’ patterns in them is the focus of network data mining methods.

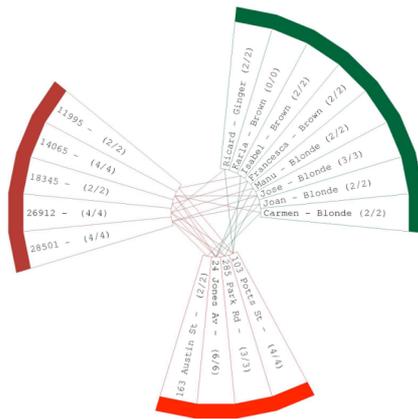
These different perspectives infer different sets of models. Fig. 2 uses the simple example in Fig. 1 to illustrate these differences. The classifier model in Fig. 2a is the well-known decision tree model (Dunham 2002). The tree shows a generalisation of the concept “student that gets sunburned” described in terms of the four input attributes in the table in Fig. 1, i.e. in terms of student height and weight, hair colour and whether they are in the habit of using lotion. The knowledge that we have discovered from the sample is that brunettes do not get sunburned regardless of whether they use lotion or not. The height and weight do not affect the result of whether you get sunburned or not. In some sense these facts are equivalent to a statistical summary of the data. As the aim is to identify general trends, the analysis and the model do not take in account the relationships between the entities described at the level of individual data points.

The structure of the relationships between the individual entities is revealed by the *network models*. The model in Fig. 2b is a social network based on the values of two columns in Fig. 1: “Name” and “Lives near”. It reveals possible relationships between Carmen, Ricard, Jose and Karla. Deeper analysis of these relationships may reveal why Carmen, Jose and Ricard are the only ones to get sunburned (e.g. it may turn out that it was Ricard’s influence to go to the beach at noon and stay there longer). On the other hand, it also reveals that there are two isolated groups of possible interest – Joan and Francesca, and Manu and Isabel (none of them got sunburned).



a. "Black-box" input-output generalization

b. Structure of a network model



c. Implicit relations between the entities linked through the values of some of their attributes. The links between students and the values of some attributes of interest ("Transaction reference #" and "Address" from Fig. 1) can reveal the existence of implicit relations.

Fig. 2. Illustration of the differences between predictive models based on generalization of the relation between the input and output attributes, and network models that take in account the relations between individual instances.

The model in Fig. 2c illustrates the implicit relations encoded through the transactions and street address. Note that this view reveals a heterogeneous network of relations between values of attributes.

Network models, which include the topology of the network and the characteristics of its nodes, links, and clusters of nodes and links, attempt to explain observed phenomena through the interactions of individual nodes or node clusters. During recent years there has been an increasing interest in these types of models in a number of research communities (see (Albert and Barabási 2002; Newman 2003) for extensive surveys of the research work on network models in different areas). Historically, sociologists have been exploring the social networks between people in different social settings. A typical social network analysis research scenario involves data collection through questionnaires or tables, where individuals describe their interactions with other individuals in the same social setting (for example, a club, school, organization, across organizations, etc.). Collected data is then used to infer a social network model in which nodes represent individuals and edges represent the interactions between these individuals. Classical social network analysis studies deal with relatively small data sets and look at the structure of individuals in the network, measured by such indices as centrality (which individuals have most links, can reach many others, are in a position to exert most influence, etc.) and connectivity (paths between individuals or clusters of individuals through the network). The body of literature is well covered by the following complementary books (Wasserman and Faust 1994; Scott 2000).

Works looking at the *discovery of network models* beyond "classical" social network analysis, date back to the early 1990s (for example, the discovery of shared interests based on the history of email communications (Schwartz and Wood 1993)). Recent years have witnessed the substantial input of physicists (Albert and Barabási 2002), mathematicians (Newman 2003) and organizational scientists (Borgatti 2003) to network research, with the focus shifting to large scale networks, their statistical properties and explanatory power, the discovery of such models and their use in explaining different phenomena in complex systems. The areas include a wide spectrum of social, biological, information, technological and other heterogeneous networks (see Table II in (Newman 2003) and Table I in (Albert and Barabási 2002), and other works of interest, (Krebs 2005) and (Batagelj and Mrvar 2003). Recent researches in the data mining community are looking at network models for predictive tasks (for example, predicting links in the model (Liben-Nowell and Kleinberg 2003), or the spread of influence through a network model (Domingos and Richardson 2001; Richardson and Domingos 2002; Kempe, Kleinberg and Tardos 2003). The interest towards link analysis and corresponding network models has increased during recent years, evidenced by the number of workshops

devoted to these topics, with a major focus on algorithms that work on graphs (for example, for example, see the presentations at recent workshops on link analysis at ACM KDD conference series¹ and SIAM Data Mining Conference²).

Such interest towards network models and new approaches to derive them has been driven largely by the availability of computing power and communication networks make possible the collection and analysis of these large amounts of data. Early social network research investigated networks of tens, at most hundreds of nodes. The networks that are investigated in different studies in present days tend to include millions (and more) links and nodes. This change of scale of the network models required change in the analytics approach (Newman 2003). Network Data Mining addresses this challenge.

However, the above mentioned efforts do not look at approaching the integrated data set and the process of facilitating discoveries from such data set. The network data mining approach is looking at this area. We define *network data mining as the process of discovering emergent network patterns and models in large and complex data sets*. The term denotes the methods and techniques in the following contexts:

- mining network models out of data sets
- mining network data (i.e. data generated by the interaction of the entities in a network, for example, in communications networks that can be the network traffic data).

The original data may not necessarily have been collected with the idea of building network models. The network patterns are derived from the integrated data set, which includes the interaction data and the descriptive attributes. Further in the chapter we briefly discuss the “loss of detail” problem and the “independency of attributes” assumption in knowledge discovery, present a human-centered knowledge discovery methodology that addresses these issues, and present a case study that illustrates the solutions that network data mining approach offers.

An illustrative comparison between predictive and network data mining is presented in Table 1.

Table 1. Comparison between predictive and network data mining

	Predictive data mining	Network data mining
Models and data	Predictive models derived from attributes data	Implicit network models derived from attributes data
Primary function	Prediction of outcomes	Discovery of irregularities
Level	Summarised data - often	Detailed elemental data
Perspective	Generalisation	Digging the details of interlinkage
Assumptions	a. Independence of attributes b. Independence of records	a. Linkage between attributes b. Linkage between records
Role of cognition	Little or none	Central and integral

The “loss of detail” problem in data mining

Data mining has been described as the art and science of teasing meaningful information and patterns out of large quantities of data – turning ‘dusty’ data that organisations have already collected into valuable information, operationally and strategically. Most data mining and analysis tools work by statistically summarising and homogenising data (Fayyad et al. 1996; Nong 2003), observing the trends and then looking for exceptions to normal behaviour. In addition, as pointed in (Fayyad 2003) “data mining algorithms are “knowledge-free”, meaning they are brittle, and in real applications lack even the very basic “common sense reasoning” needed to recover even from simple situations. This process results in a *loss of detail* which, for intelligence and detection work, can defeat the whole purpose as it is often in the detail where the most valuable information is hidden. More generally, the identifying of exceptions to the ‘norm’ requires a top down-approach in which a series of correct assumptions needs to be made about what is normal and abnormal, and what will be applied to constitute a query. For many complex problems it can be difficult to even start this process since it is impossible to be specific about normal behaviour and what could or should constitute an exception.

The “independency of attributes” assumption in data mining

The “independency of attributes” assumption is accepted in some forms of data mining (for example, classifiers building algorithms like Naïve Bayes (Dunham 2002)). Under this assumption, the distributions of the values of the attributes are independent of each other. However, real-world data rarely satisfies the attribute value independence assumption. In fact, some data mining techniques like association and correlation analysis (Han and Kamber 2001), techniques that look at causality and the discovery causal networks (for example, Bayesian network models (Ramoni

¹ <http://www-2.cs.cmu.edu/~dunja/LinkKDD2004/>

² <http://www-users.cs.umn.edu/~aleks/sdm04w/>

and Sebastiani 2003)), make exactly the opposite assumption. Moreover, there are situations where the assumption sounds counterintuitive as well. For example, it is natural for the salary value to be correlated with the values of the age in a sample.

The logic is clear: by missing detail or making the wrong assumptions or simply by being unable to define what is normal, an organisation that relies solely upon predictive data mining may fail to discover critical information buried in its data.

Network data mining – the methodology

When there are few leads and only an open-ended specification on how to proceed with an analysis, *discovery* is all important. The discovery phase in the analysis process is too often overlooked or only implemented via exception based detection methods that are constrained to domain knowledge already known. Increased needs for automated detection have been accompanied by an increased reliance upon exception based forms of detection and rules based querying. However, along with the proliferation of data and increasingly advanced concealment tactics employed by the parties that want to avoid detection, there is a dilemma. We cannot write the rules ahead of time to specify a query or to write an exception (e.g. for an outlier, a threshold, an alert or an alarm) if the nature of what constitutes an exception and therefore the ability to specify relevant rules is unknown.

Network data mining is concerned with *discovering* relationships and patterns in linked data, i.e. the inter-dependencies between data items at the lowest elemental level. These patterns can be revealing in and of themselves, whereas statistically summarised data patterns are informative in different but complementary ways.

Similar to aspects of visual data mining (Wong 1999), network data mining integrates the exploration and pattern spotting abilities of the human mind with the processing power of computers to form a powerful knowledge discovery environment that is supposed to capitalise on the best of both worlds. This human-centred approach creates a powerful solution. However, to realise its full value the discovery phase needs to be repeated at regular intervals so that new irregularities that arise and variations on old patterns can be identified and fed into the *exception detection phase*.

The overall network data mining process is illustrated in Fig. 3. In network data mining, the data miner is in a role similar to Donald Schön's "reflective practitioner" (Schön 1983; Schön 1991), originally developed in the analysis of design processes. In his later work (Schön, 1991), Schön has presented a number of examples of disciplines that fit a process model with characteristics similar to the design process. What is relevant to our claim is Schön's view that designers put things together (in our case, the miner replaces the designer, and the things s/he puts together are the visual pieces of information) and create new things (in our case the miner creates new chain of inquiries interacting with the views) in a process involving large amount of variables (in our case these are the attributes and the linkages between them). Almost anything a designer does, involves consequences that far exceed those expected. In network data mining approach the inquiry techniques may lead to results that far exceed those expected and that in most cases may change the line of the analysis of the data into an emerging path. Design process is a process which has no unique concrete solution. In network data mining we operate with numerous network slices of the data set, assisting in revealing the different aspects of the phenomena. Schön also states that he sees a designer as someone who changes an indefinite situation into a definite one through a reflexive conversation with the material of the situation. By analogy, the network data miner serves to change the complexity of the problem through the reflexive investigative iterations over the integrated data set. The reflective step is a revision of the reference framework taken in the previous step in terms of attributes selection, set of visual models and corresponding guiding techniques, and the set of validation techniques.

The main methodological steps and accompanying assumptions of network data mining (NDM) approach include:

Sources of data and modelling: NDM provides an opportunity to integrate data so as to obtain a single address space, a common view, for disparately sourced data. Hence, a decision as to which sources are accessible and most relevant to a problem is an initial consideration. Having arranged and decided the sources, the next question relates to modelling. Which fields of data should serve as entities/nodes (and attributes on the entities) and which should serve as links (and attributes on the links)³. Multiple data models can and often are created to address a particular problem.

Visualization: The entities and a myriad of linkages between them must be presented to screen in meaningful and color coded ways so as to simplify and facilitate the discovery of underlying patterns of data items that are linked to other items and on-linked to still other items. This is especially so with large volumes of data, e.g. many hundreds of thousands or millions of links and entities which the user must be able to easily address and then readily make sense of from an interpretative and analytical point of view.

³ A tool that interfaces to relational databases and any original sources of data (e.g. XML files) is basic to NDM, and is a capability provided in the NetMap software suite which is a premier technology in this space and was used in this case presented later in this chapter.

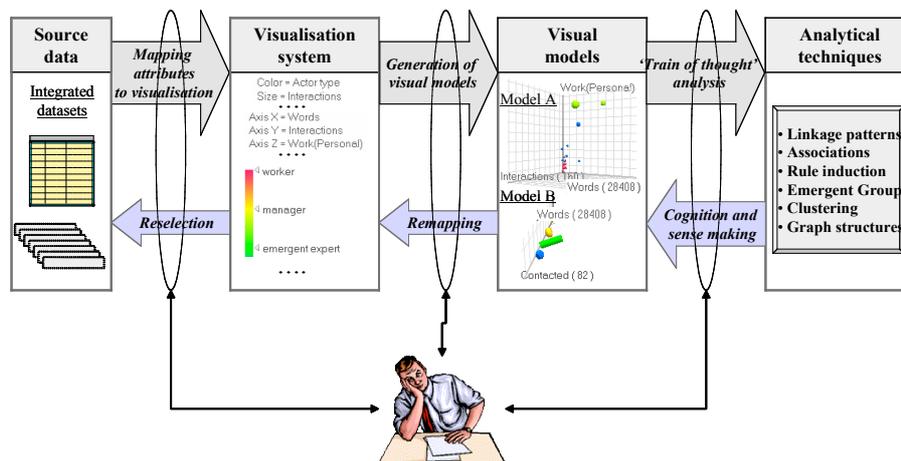


Fig. 3. Network data mining as a human-centered knowledge discovery process.

The main methodological steps and accompanying assumptions of network data mining (NDM) approach include:

Sources of data and modelling: NDM provides an opportunity to integrate data so as to obtain a single address space, a common view, for disparately sourced data. Hence, a decision as to which sources are accessible and most relevant to a problem is an initial consideration. Having arranged and decided the sources, the next question relates to modelling. Which fields of data should serve as entities/nodes (and attributes on the entities) and which should serve as links (and attributes on the links)⁴. Multiple data models can and often are created to address a particular problem.

Visualization: The entities and a myriad of linkages between them must be presented to screen in meaningful and color coded ways so as to simplify and facilitate the discovery of underlying patterns of data items that are linked to other items and on-linked to still other items. This is especially so with large volumes of data, e.g. many hundreds of thousands or millions of links and entities which the user must be able to easily address and then readily make sense of from an interpretative and analytical point of view.

'Train of thought' analysis: Linkage between data items means that the discovery of patterns can be a process whereby the analyst uses the reflective practitioner approach mentioned earlier. Explicit querying is less often the case; rather the analyst may let the intuition guide him or her. For example, "Why are all those red links going over there?", "What are they attached to, and in turn what are they attached to?" Such 'train of thought' processes invariably lead the analyst to discover patterns or trends that would not be possible via more explicit querying or exception based approaches – for the specification for the queries is not known.

Cognition and sense-making: An integral assumption in NDM is that the computer in the analyst's mind is more powerful by orders of magnitude that the one on the desktop. Hence, intuition and cognition are integral, and need to be harnessed in the analytical process especially at the discovery phase in those cases where there is only limited domain knowledge as a guide to analysis and understanding.

Discovery: An emergent process, not prescriptive one. It is not possible to prescribe ahead of time all the query rules and exception criteria that should apply to a problem, if domain knowledge is less than perfect. And of course in many if not most cases it is, otherwise the problem would already have been solved. By taking an emergent or bottom up approach to what the data are 'saying', patterns and linkages can be discovered in a way that is not too different from 'good old fashioned' policing, where curiosity and intuition have always been integral in the ability to discover the facts, then qualifying them and solving the crime.

Finding patterns that can be re-discovered: Any linkage pattern observed on screen is simply that, an observation of potential interest. In the context of retail NDM for example, any sales assistant with a high ratio of refunds to sales (statistically flagged) might attract attention. In a case in point known to the authors, the perpetrators of a scam knew about such exception criteria. As longer term employees "in the know", they could easily duck under them. They had taken it in turns to report levels of refunds always just under the limits no matter what the limits were varied to over an extensive period. NDM was able to show collusive and periodic reporting linkages to supervisors – patterns discovered through visualization and algorithms that facilitate the intuition. Such patterns are of particular interest, and in fact often an objective of the network mining approach. They are termed *scenarios* and characterised as definable and re-usable patterns. Their value is that they are patterns that have now become 'known'. Hence they can be defined, stored in a knowledge base, and applied at the front end of a process as, for example, in a predictive modelling environment. The important methodological step is that the definitions need to be discovered in the first place before they can be further applied.

Network data mining is complementary to statistical summarizing and exception detection data mining. Network data mining is particularly useful in the *discovery phase*, of finding things previously unknown. Once discovered, particular patterns and abnormal behaviors and exceptions are of course able to be better defined, and these scenarios

⁴ A tool that interfaces to relational databases and any original sources of data (e.g. XML files) is basic to NDM, and is a capability provided in the NetMap software suite which is a premier technology in this space and was used in this case presented later in this chapter.

can then be saved and applied automatically across new volumes of data, including by the feeding of discovered scenarios back into traditional data mining tools. The section below illustrates the network data mining approach using a real-world case study

Example of network data mining approach in fraud detection

Many cases could be used to illustrate a network data mining approach and each case would be instructive of different features. Nonetheless, this particular case is not atypical of the methods involved in knowledge discovery in network data mining. The case is presented as a reflection on the steps that the analyst did. The presentation is structured along the main methodological steps that we have identified in the previous section.

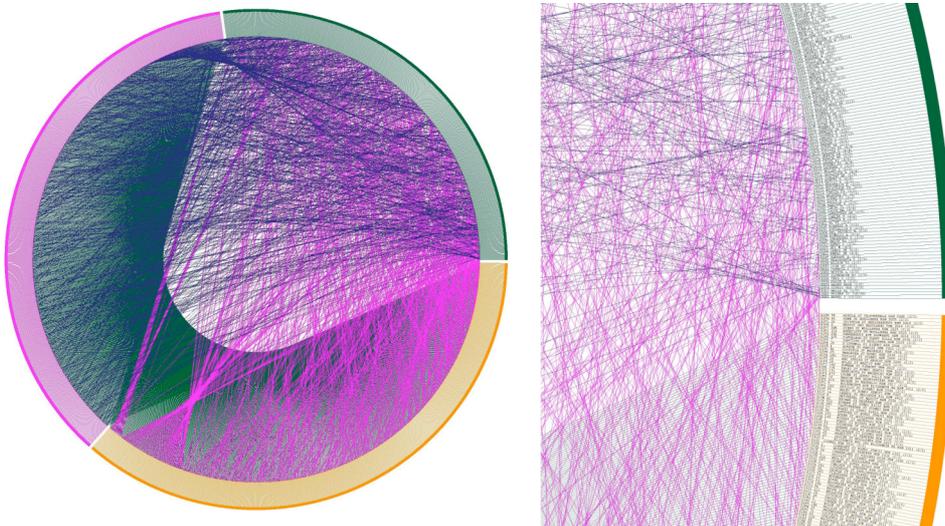
Sources of data and modelling: The case involved analysis of approximately twelve months of motor vehicle insurance claims from one company. A small set of five thousand records were available as a pilot project from one state of Australia. Note that all the information presented here has been de-identified.

The brief from the company was essentially open-ended and without specification. A concern was expressed that there *could* be suspicious transactions and perhaps fraudulent activity but there were no known persons or transactions of interest. The case was clearly in the realm of ‘discovery’.

Visualization: The study used the NetMap software from NetMap Analytics. The analyst first built a set of linkages from the available fields. These were between persons, addresses, claim numbers, telephone numbers, and bank accounts into which claims monies had been paid.

Initially the analyst only looked at three fields of data and the linkages identified between them, as shown in Fig. 4. To start to make sense of the data overviewed in Fig. 4, the analyst then processed the data through an algorithm in NetMap to produce the display of potential irregularities shown in Fig. 5 (also close-ups in Fig. 6).

Cognition and sense-making (and ‘Train of thought’ analysis): What appeared to be ‘regular’ patterns were observed. These were seen to be small triangles of data comprised of a person, a claim number and an address, all fully inter-linked. The explanation as to why the little triangles appeared to be ‘regular’ patterns in the data was simple after the analyst had stopped to think about it: most people just had one claim and one address. In comparison, the ‘bumps’ looked as though they were ‘irregularities’. The ‘bumps’ comprised persons linked to multiple claims and/or addresses. By taking the seemingly regular patterns out of the picture (the triangles), the analyst was quickly able to produce a short list of potentially suspicious transactions and inter-related behaviours. That is, from a larger amount of data she was able to quickly focus *where* to look in the myriad of linkages to drill down for more details.



- a. Overview of links between persons, addresses and claim numbers
- b. Close up at approximately 3 o'clock of the linked data shown in Fig. 4a.

Fig. 4. An illustration of an initial macro view of linkages in the data with a subsequent step in digging deeper in the details

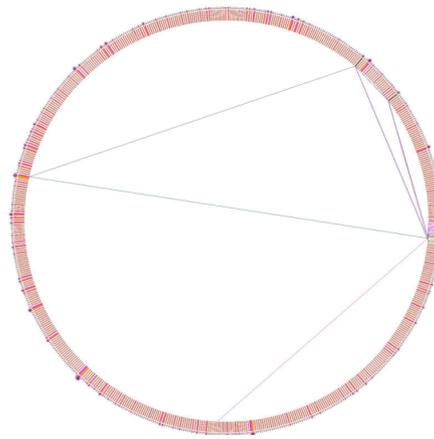
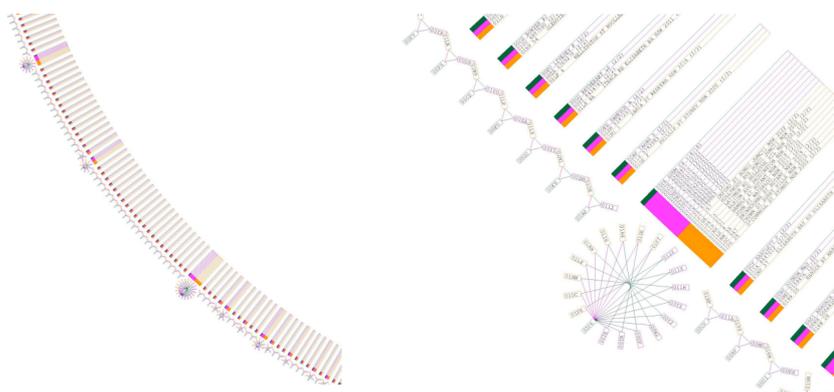


Fig. 5. Data from Fig. 4 processed to show certain patterns of irregularities (see detail in Fig. 6)



- a. Overview of links between persons, addresses and claim numbers
- b. Overview of links between persons, addresses and claim numbers

Fig. 6. Close up segments from approximately 4 o'clock in Fig. 5 showing what appeared to be regular patterns (the small triangles of linkage) and also irregularities (the larger 'bumps')

File View		WESSON E / 93845152					93845152							
Attributes	Connections	Map	Named groups	Properties	Direction	Link Type	Reference	Date	Amount	Attributes	Connections	Map	Named groups	Prop
Node Id	000H				→→	PERSON - PHONE NUMBER	HURCR8809604	03-Jun-1998	5325	Node Id	3221			
Node Name	WESSON E				→→	PERSON - PHONE NUMBER	CRK0124130	05-Jun-1998	4525	Node Name	93845152			
Node Type	PERSON				→→	PERSON - PHONE NUMBER	CR89009370	01-Mar-1998	4200	Node Type	PHONE NUMBER			
Suburb					→→	PERSON - PHONE NUMBER	CR89009370	23-Jun-1997	850	Suburb				
Employee	0				→→	PERSON - PHONE NUMBER	CR88092738	12-May-1998	2300	Employee	0			
Node Number	592									Node Number	0			
Node Link Count	32									Node Link Count	0			

Fig. 7. More detailed information underlying any 'entity' or 'link' was able to be accessed by clicking on that data element. The analyst could then quickly qualify observed patterns.

Discovery (and 'Train of thought' analysis): Other and more interesting types of 'irregular' patterns in this case were those seen to be extending across the middle of the main circle in Fig. 5. It was clear to the analyst that most of the data items did not have links across the middle and, hence, these were 'irregularities' of some sort. They seemed to emanate from about the 3 o'clock position in Fig. 5. Accordingly, the analyst zoomed into the display at about 3 o'clock and selected one of the inter-linked data items for 'step-link' purposes. She selected Simons JL, but it could have been any of these data items since several were linked to each other. The beginning of this step is shown below in Fig. 8.

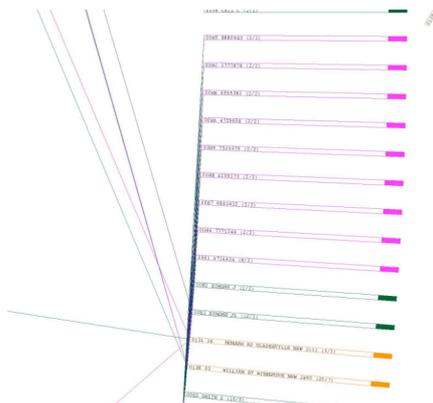


Fig. 8. The analyst chose one of these inter-linked data items to ‘step out’ from (Simons JL)

Finding patterns that can be re-discovered: Step-linking from a selected data item follows the network links out through any number of degrees of separation. As shown in Fig. 9 and Fig. 10, the analyst stepped out from JL Simons to ‘infinity’ degrees of separation, the net effect being to bring to screen all of that party’s indirect linkage. Obviously, if further sources of data could have been added (which is often done but additional sources were unavailable in this case) richer indirect linkages would have most likely further assisted the inquiry. Also, although the analyst did not do so in this case, destinations of data items may be specified and then stepped to from chosen source data items, thereby determining whether certain parties are linked at all and if so who or what are the intermediaries.



Fig. 9. Step-linking from the selected entity to infinity degrees of separation

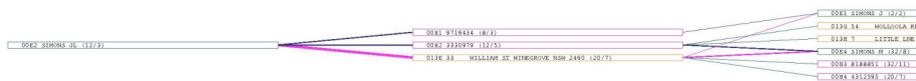


Fig. 10. Close up of portion of the data shown in Fig. 9.

The particular extract of linked data in Fig. 9 and Fig. 10 was then processed through an algorithm unique to NetMap called *emergent groups*. The resulting pattern is shown in Fig. 11.

Discovery: From the extract of data in Fig. 11, four emergent groups were identified as shown in Fig. 12. An emergent group comprises closely inter-related data items; they have more links within the group than outside to any other group. Thus, they are defined ‘out of’ the data in a bottom-up way (‘what are the data telling us?’), rather than prescriptively by rules or queries which in this case would have been impossible - there was simply no prior knowledge about how to frame such rules or program any queries. The analyst was squarely in ‘discovery’ mode.

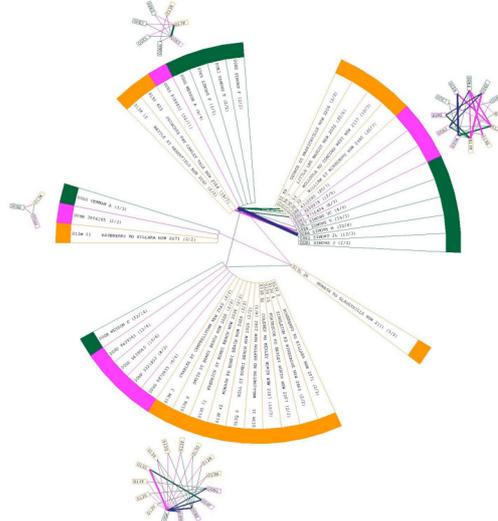


Fig. 11. Emergent groups discovered within the extract of data shown in Fig. 9

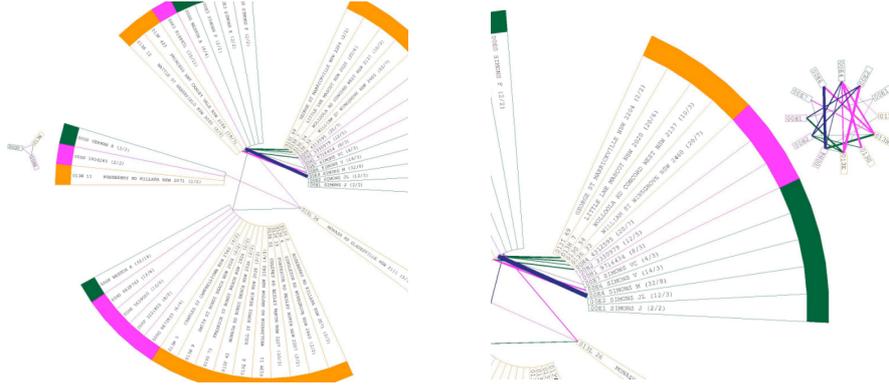


Fig. 12. Close-up views of the emergent groups shown in Fig. 11

‘Train of thought’ analysis: The emergent group on the right in Fig. 12 comprised five people called Simons, three claims and four addresses, all closely inter-related (relationships between id codes are shown in the satellite, and the id codes plus full names are shown in an the arc on the main circle). They were linked across to another group at 11 o’clock comprised of more people called Simons and somebody called Wesson. That Wesson (initial A) was linked down to the group at 5 o’clock to E Wesson via a common claim. That in turn took the analyst over to the address at 4 o’clock and then to an ‘A Verman’ at 9 o’clock.

This ‘train of thought’ analysis led the analyst to Verman. Her intuition indicated she wanted to look at Verman more closely although she could not have been specific as to why. To cut a long story short, when Verman was investigated and went to jail it was learned that he had originally had a ‘regular’ pattern (comprising one claim and one address – see Fig. 6). He reckoned this was a good way to make money. So, he recruited the Simons and the Wessons as the active parties in a scam of staged vehicle accidents while he tried to lie as low as he could. He was careless however, since he had left a link to another address (the one at 4 o’clock in Fig. 11 – note, he must have been a witness or a passenger). This link indirectly implicated him back into the activity that involved the Simons and Wessons.

Finding patterns that can be re-discovered: The analyst did not know this of course at the time, but could sense that Verman was a few steps removed from the activity of the Simons and thought she would like to quickly qualify this person. She firstly took Verman and stepped out to obtain all his indirect linkage (see Fig. 13).

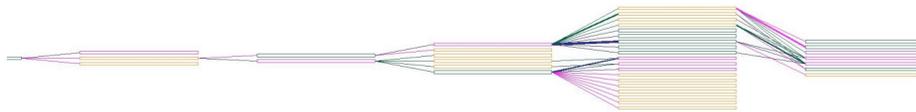


Fig. 13. A potential suspect on the left (Verman) with all his indirect linkage to other data items

The analyst then sought to ‘enrich’ the linkage of Verman, i.e. to regard him as a potential case that she would like to qualify on the spot if possible, by adding extra linkage (as shown in Fig. 14). In this case, she only had two extra fields available: bank account information and telephone numbers. Nonetheless, she quickly discovered one extra and crucial link that helped her to qualify Verman – one of his two telephone numbers was also linked to A Wesson (see Fig. 15). That additional link provided the ‘tipping point’, the extra knowledge that gave her sufficient confidence to recommend that Verman be investigated. This subsequently led to his arrest and conviction on fraud charges.



Fig. 14. Enrichment of the linkage in Fig. 13 by adding in extra fields of data.

It transpired that Verman had been the mastermind behind a claims scam totalling approximately \$150,000. He could not have been discovered by traditional data mining methods since no domain knowledge existed that could help define any exception rules or serve as inputs to SQL queries. If red flags or alerts had been applied as the only investigative approach, he would have escaped detection since he was essentially ‘regular’ and not unusual in anyway. The use of detailed linkage and the easy ability to navigate and make sense of it made the difference.



Fig. 15. Close up of portion of Fig. 14 showing the extra ‘tell tale’ link (arrowed: a telephone number in common). This extra information led to an investigation and then the arrest and successful prosecution of Verman, the person shown on the left.

Conclusion

This chapter has described the concept of network data mining and presented a case study by way of illustrating its real-world implementation and its distinction from more traditional approaches to data mining, and also its distinction from social network analysis.

Network data mining focuses upon knowledge discovery. It involves a human-centred process which harnesses the intuitive powers of the human intellect in conjunction with color-coded linkage patterns and unique algorithms to facilitate the intuition, a process referred to as ‘train of thought’ analysis.

We can summarize the main steps in the knowledge discovery process (Fayyad et al. 1996) as follows:

1. Define scenarios in terms of query specifications and exception rules
2. Process the data
3. Interpret or initiate action

The discovery phase, which network data mining gives emphasis to, logically precedes and feeds into step 1. This prior step we refer to as step 0:

0. Discover patterns and qualify them as scenarios.

Note also that discovery (step 0) and exception detection (step 1) are inter-related. Once patterns of interest have been discovered then they can be defined and this information incorporated in more conventional exception detection and querying. However, many situations require that discovery takes place first. It then needs to be applied to the same sets of data and new sets since endless variations and inventive ways of concealing nefarious activity and avoiding detection are put into play that need to be discovered in order to keep ahead of the game.

We presented a case which illustrated the cornerstones of the network data mining approach. The party in the case who was eventually convicted (Verman) would have escaped detection if a traditional data mining approach had been used. He had only had one claim, no ‘red flag’ information was involved, and nothing particularly anomalous occurred with respect to him. By all accounts he would have slipped under any exception detection processes.

Successful discovery did occur through the use of the network data mining methods outlined. Train of thought analysis enabled a discovery to be made that would have been highly unlikely otherwise. Querying could have not have been successful since there is no way that a relevant query could have been framed. Exception rules could not have been specified since there was no information as to what could constitute an exception that would have discovered Verman. He was essentially below the radar.

Masterminds concealing their behavior, tend to be as unobtrusive as possible. They also often know the rules and the exceptions and know what they need to do if the rules and exceptions change so as to avoid being caught. Hence, any discovery tool must go beyond programmatic and prescriptive exception based detection.

A complementary usage of traditional data mining could have in principle been used in this case as follows. Several of the underlings recruited by Verman had the same family name, Simons. Hence, the name Simons appeared more often than expected in the telephone directory. Therefore, a rule could have been to display all names occurring more often than expected in the telephone population. The result being that the name Simons would be flagged. In a complementary approach, this flagging would be a helpful initial task. The next task would be within an NDM linked data environment to ‘step out’ several steps from the Simons and so commence the discovery process from that point.

This twin approach has been used to great effect. It essentially uses statistical approaches to flag where the detailed linkage should be examined by discovery-oriented ‘train of thought’ analysis. In numerous cases we are aware of this combined approach has leveraged the best of both network and non-network data mining.

References

- Albert, R. and A.-L. Barabási (2002): "Statistical mechanics of complex networks." *Reviews of Modern Physics* 74(January 2002), 47-97.

- Antonie, M.-L., O. R. Zaiane, et al. (2003): Associative classifiers for medical images. *Mining Multimedia and Complex Data*. O. R. Zaiane, S. J. Simoff and C. Djeraba. Heidelberg, Springer, 68-83.
- Batagelj V. and A. Mrvar (2003): *Pajek - Analysis and Visualization of Large Networks*. In: Inger M and Mutzel P, eds. Graph Drawing Software. Berlin: Springer, 2003.
- Borgatti, S. P. (2003): "The network paradigm in organizational research: A review and typology." *Journal of Management* **29**(6), 991-1013.
- Domingos, P. and M. Richardson (2001): Mining the network value of customers. *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, ACM Press, 57-66.
- Dunham, M. H. (2002): *Data Mining: Introductory and Advanced Topics*, Prentice Hall.
- Fayyad, U. M. (2003): "Editorial." *ACM SIGKDD Explorations* **5**(2), 1-3.
- Fayyad, U. M., G. Piatetsky-Shapiro, et al. (1996): From data mining to knowledge discovery: An overview. *Advances in Knowledge Discovery and Data Mining*. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy. Cambridge, Massachusetts, AAAI Press/The MIT Press, 1-34.
- Han, J. and M. Kamber (2001): *Data Mining: Concepts and Techniques*. San Francisco, CA, Morgan Kaufmann Publishers.
- Hand, D., H. Mannila, et al. (2001): *Principles of Data Mining*. Cambridge, Massachusetts, The MIT Press.
- Kempe, D., J. Kleinberg, et al. (2003): Maximizing the spread of influence through a social network. *Proceedings ACM KDD2003*, Washington, DC, ACM Press
- Krebs, V. (2005): <http://www.orgnet.com>
- Klösigen, W. and J. M. Zytow, Eds. (2002). *Handbook of Data Mining and Knowledge Discovery*, Oxford University Press.
- Liben-Nowell, D. and J. Kleinberg (2003): The link prediction problem for social networks. *Proceedings CIKM'03*, November 3–8, 2003, New Orleans, Louisiana, USA., ACM Press
- Newman, M. E. J. (2003): "The structure and function of complex networks." *SIAM Review* **45**, 167-256.
- Nong, Y., Ed. (2003). *The Handbook of Data Mining*. Mahwah, New Jersey, Lawrence Erlbaum Associates.
- Nong, Y. (2003): Mining computer and network security data. *The Handbook of Data Mining*. Y. Nong. Mahwah, New Jersey, Lawrence Erlbaum Associates, 617-636.
- Ramoni, M. F. and P. Sebastiani (2003): Bayesian methods for intelligent data analysis. *Intelligent Data Analysis: An Introduction*. M. Berthold and D. J. Hand. New York, NY, Springer, 131-168.
- Richardson, M. and P. Domingos (2002): Mining knowledge-sharing sites for viral marketing. *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining*, Edmonton, Canada, ACM Press, 61-70.
- Schön, D. (1983): *The Reflective Practitioner*. New York, Basic Books.
- Schön, D. (1991): *Educating The Reflective Practitioner*. San Francisco, Jossey Bass.
- Schwartz, M. E. and D. C. M. Wood (1993): "Discovering shared interests using graph analysis." *Communications of ACM* **36**(8), 78-89.
- Scott, J. (2000): *Social Network Analysis: A Handbook*. London, Sage Publications.
- Scott, J. (2000): *Social Network Analysis: A Handbook*. London, Sage Publications.
- Wasserman, S. and K. Faust (1994): *Social Network Analysis: Methods and Applications*. Cambridge, Cambridge University Press.
- Weiss, S. M. and T. Zhang (2003): Performance analysis and evaluation. *The Handbook of Data Mining*. Y. Nong. Mahwah, New Jersey, Lawrence Erlbaum Associates.
- Wong, P. C. (1999): "Visual Data Mining." *IEEE Computer Graphics and Applications* **September/October**, 1-3.

CONTRIBUTED PAPERS

The Formal Semantics of the TimeER Model

Heidi Gregersen

Aarhus School of Business
Fuglesangs Allé 4, DK-8210 Aarhus V, Denmark,
Email: hgr@asb.dk

Abstract

A wide range of database applications manage information that varies over time. Many of the underlying database schemas of these were designed using one of the several versions of the Entity-Relationship (ER) model. In the research community as well as in industry, it is common knowledge that the temporal aspects of the mini-world are pervasive and important, but are also difficult to capture using the ER model. Not surprisingly, several enhancements to the ER model have been proposed in an attempt to more naturally and elegantly support the modeling of temporal aspects of information. Common to most of the existing temporally extended ER models is that the semantics of the models are unclear. This problem is addressed in this paper by developing a formal semantics for the TIMEER model based on denotational semantics.

Keywords: Conceptual modeling, database design, entity-relationship models, temporal databases, temporal data models, temporal semantics.

1 Introduction

A wide range of prominent, existing database applications manage time-varying information. These include financial applications such as portfolio management, accounting, and banking; record-keeping applications, including personnel, medical-record, and inventory; and travel applications such as airline, train, and hotel reservations and schedule management.

Frequently, existing temporal-database applications such as these employ the Entity-Relationship (ER) model [2], in one of its different incarnations, for database design. The model is easy to understand and use, and an ER diagram provides a good overview of a database design. The focus of the model is on the structural aspects of the mini-world (we use the term “mini-world” for the part of reality that the database stores information about), as opposed to the behavioral aspects. This focus matches the levels of ambition for documentation adopted by many users.

In the research community as well as in industry, it has been recognized that although temporal aspects of mini-worlds are prevalent and important for most applications, they are also difficult to capture elegantly using the ER model. The temporal aspects have to be modeled explicitly in the ER diagrams, resulting in ER diagrams with entities and

attributes that model the temporal aspects and that make otherwise intuitive and easy-to-comprehend diagrams difficult to understand. As a result, some industrial users simply ignore all temporal aspects in their ER diagrams and supplement the diagrams with textual phrases such as “full temporal support,” indicating that the temporal aspects of data should somehow be captured. The result is that the ER diagrams do not document well the temporally extended relational database schemas used by the application programmers.

The research community’s response to the shortcomings of the regular ER model for the modeling of temporal aspects has been to develop temporally enhanced ER models, and a number of models have been reported in the research literature. These temporal ER models are developed in an attempt to provide modeling constructs that more naturally and elegantly enables the designer to capture temporal aspects, such as valid and transaction time, of information. For a detailed description of some of the existing models, see Gregersen and Jensen [10] where 11 models are examined [4, 6, 7, 15, 16, 17, 19, 20, 22, 23, 24]. For details on more recent models, see [1, 9, 11, 18].

The approaches taken to add built-in temporal support into the ER model are quite different. The temporal ER models generally either change the semantics of the existing ER model constructs or introduce new constructs to the model. One approach is to devise new notational shorthands that replace some of the patterns that occur frequently in ER diagrams when temporal aspects are being modeled. Another approach is to change the semantics of the existing ER model constructs, making them temporal. In its extreme form, this approach does not result in any new syntactical constructs—all the original constructs have simply become temporal.

A common characteristic of the existing temporal ER models for the TIMEER model, is that few or no specific requirements to the models were given by their designers. Rather than being systematically founded on an analysis of general concepts and temporal aspects, their designs are often ad hoc. For example, the design of one model is the result of the need for the modeling of temporal aspects in a specific application [7]. The definitions of the existing models also generally lack comprehensiveness and precision and rely heavily on the reader’s intuition. These conditions make it difficult to identify the ideas behind the designs of the models and to understand their semantics. Understanding the semantics of a model is especially important for database designers when they have to determine which model to choose for a specific database design.

Most of the proposed models define the semantics of the models in terms of the relational model where the relations are augmented with time attributes that record the specified temporal aspects

Copyright ©2006, Australian Computer Society, Inc. This paper appeared at the Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 53. Markus Stumptner, Sven Hartmann, and Yasushi Kiyoki, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

of entities, relationships, and attributes. The designers of the model therefore redefine the conventional algorithm that transform an ER diagram into a relational database schema. Another approach taken is to convert the temporal construct of the temporal ER model into conventional ER constructs [24] and then reuse the conventional transformations algorithm and define the semantics of their new constructs in terms of the relational model. This approach is perfectly fine if the newly designed temporal database is to be implemented on a relational platform, but with other platforms emerging, e.g, XML databases semantics defined in terms of the relational model is not sufficient.

In this paper we define a formal semantics for temporally extended ER models, based on denotational semantics. The semantics have been developed for the TIMEER model [11], but can be applied to other temporally extended models. This is true due to the fact that we first develop a textual representation, in Backus Naur Form, of TIMEER diagrams and then determine the semantics based in the textual representation. Diagrams produced by any of the other existing models can be translated into the textual representation of diagrams presented in this paper. We have chosen to develop the formal semantics for this particular model because the description of this model is clear and easy to follow.

The paper is structured as follows. Section 2 provide a presentation of the TIMEER model. Section 3 gives the formal semantics of the model. Section 4 concludes and identifies promising research directions.

2 The TimeER Model

In this section we provide a short introduction to the Time Extended ER (TIMEER) model as it is defined in a paper by Gregersen and Jensen [11]. Furthermore an introduction to the running example of this paper is given.

The TIMEER model extends the EER model as defined by Elmasri and Navathe [5] to provide built-in support for capturing temporal aspects of entities, relationships, superclasses and subclasses, and attributes. The design of the model is based on an ontology, which defines database objects, fundamental aspects of time, and indicates which aspects of time may be associated meaningfully with which database objects. Next, the model is designed to satisfy additional, explicitly formulated design goals for temporally extended ER models.

Database Objects

First we present and explain the database objects of the TIMEER model. Anything that exists in the mini-world and can be separated from other things in the mini-world is an *entity*; hence, a data model used for capturing a database representation of an entity should provide means of conveniently modeling the existence and unique identification of entities. The time during which an entity exists in the mini-world, that is, the time during which it is of interest to the mini-world we call the *existence time* of the entity. Other models have a different view of existence time [4, 6].

Beyond having an independent existence, an entity is characterized by its properties, modeled by attributes. At any given point in time, an entity has a value for each of its attributes. The values of some attribute remain unchanged over time while others vary, that is, at different points in time, the values of an attribute for an entity may be different. We assume

that it is meaningful for entities to have properties exactly when they exist (i.e., when they are entities)—it is meaningless for something that does not exist to have properties.

A database represents sets of entities that are similar, that is, have the same structure, or put differently, entities that have the same attributes. Entity types define sets of entities with the same attributes, and the entities of the same type is termed an entity set.

Entities may be interrelated via relationships. Such relationships can be seen from two very different points of view. We can either perceive relationships among entities as attributes of the participating entities, or we can perceive relationships as having existence in their own right. Both points of view have merit.

A relationship type among some entity types defines a set of relations among entities of these types. Each relationship relates exactly one entity from each of the entity types that the relationship type is defined over. The set of relationships defined by a relationship type is called a relationship set.

Another type of relationships exists, namely the superclass/subclass relationships that classify entities of a superclass into different subclasses, e.g., employees may be divided into secretaries, engineers, and technicians. It is the same entities that occur in the subclasses and in the superclass; superclass/subclass relationships represent inheritance hierarchies rather than relate entities. For this reason, superclass/subclass relationships cannot exist in their own right, and nor can they be seen as attributes of the involved entity types. The entities of the subclasses inherit all the properties of entities of the superclass. It is not possible in subclasses to delete or modify the inherited properties, but it is possible to add new properties.

A data model should make it possible to conveniently and concisely capture all information about the mini-world that is meaningful to capture and is relevant for the application at hand. For example, since entities exist during some periods of time, it should be possible to capture this in the data model. In turn, this implies that the database designers should have the ability to indicate, using the conceptual data model, whether or not they want to register these periods of time for the entities.

2.1 The Temporal Aspects Supported

In this section we present and explain the temporal aspects of data that the TIMEER model supports. In the database community, several types of temporal aspects of information have been discussed over the years. The model supports four distinct types of temporal aspects that are candidates for being given built-in support in an ER model, namely *valid time*, *lifespan*, *transaction time*, and *user-defined time* [12].

We use the term “fact” to denote any statement that can be assigned a truth value, i.e., true or false. The notion of *valid time* applies to facts: the valid time of a fact is time when that fact is true in the mini-world. Thus, any fact in the database may be associated with a valid time. However, the valid time may or may not be captured explicitly in the database.

In ER models, unlike in the relational model, a database is not structured as a collection of facts, but rather as a set of entities and relationships with attributes, with the database facts being implicit. Thus, the valid times are associated only indirectly with facts. As an example consider an Employee entity “E1” with a Department attribute. A valid time of June 1996 associated with the value “Shipping” does not say that “Shipping” is valid during June 1996, but

rather than the fact “*E1 is in Shipping*” is valid during June 1996. Thus, when valid time is captured for an attribute such as Department, the database will record the varying Department values for the Employee entities. If it is not captured, the database will (at any time) record only one department value for each Employee entity.

The *lifespan* of an entity captures the existence time of the entity [12]. If the concept of lifespan of entities is supported, this means that the model has built-in support for capturing the times when entities exist in the mini-world. The lifespan of an entity e may be seen as the valid time of the related fact, “*e exists.*” However, we choose to consider lifespans as separate aspects since the recording of lifespans of entities is important for many applications. If relationships are regarded as having existence in their own right, the concept of lifespan is also applicable to relationships, with the same meaning as for entities.

The *transaction time* of a database fact is the time when the fact is current in the database and may be retrieved. As is the case for lifespans, the transaction time of a fact F may be seen as the valid time of a related fact, namely the fact, “*F is current in the database,*” but we have also chosen to record transaction time as a separate aspect. Unlike valid time, transaction time may be associated with any element stored in a database, not only with facts. Thus, all database elements have a transaction-time aspect.

Observe that all the above-mentioned temporal aspects have a duration.

User-defined time is supported when time-valued attributes are available in the data model [21]. These are then employed for giving temporal semantics—not captured in the data model, but only externally, in the application code and by the database designer—to the ER diagrams. For employee entities, such attributes could record birth dates, hiring dates, etc.

We are now ready to present the graphical notation of the model. Figure 1 presents a TIMEER diagram modeling a company database. The annotations added to the modeling construct are used to indicate which temporal aspects are to be captured. The annotations are LS, indicating lifespan support, VT indicating valid-time support, TT indicating transaction-time support, LT indicating lifespan and transaction-time support, and BT indicates valid- and transaction-time support.

Example 2.1 The TIMEER diagram in Figure 1 is a diagram modelling a company divided into different departments. Each department has a number, a name, some locations, and is responsible for a number of projects. The company keeps track of when a department is inserted and deleted. It also keep track of the various locations of a department. A department keeps track of the profits it makes on its projects. Because the company would like to be able to make statistics on its profits, each department must record the history of its profits over periods of time.

Each project has a manager who manages the project and some employees who work for the project. Each project has an ID and a budget. The company registers the history of the budget of a project. Each project is associated with a department that is responsible for the project. Each employee belongs to a single department throughout his or her employment. For each employee, the company registers the ID, the name, the date of birth, and the salary. The company also records the history of employments. The departments would like to keep records of the different employees’ salary histories. For reasons of accountability, it is important to be able to trace previous records of both profits and salaries.

Employees work on one project at a time, but employees may be reassigned to other projects, e.g., due to the fact that a project may require employees with special skills. Therefore, it is important to keep track of who works for which project at any given point in time and when they are suppose to be finished working on their current project. Some of the employees are project managers. Once a manager is assigned to a project, the manager will manage the project until it is completed or otherwise terminated.

3 Formal Semantics of TIMEER

This section defines the formal semantics of TIMEER. As a first step, we translate the graphical TIMEER diagram into an equivalent textual representation which can be seen as a database schema. The semantics of a TIMEER diagram is then defined as the semantics of the equivalent textual variant of the diagram.

In Section 3.1, we present the textual representation of the model and exemplify the transformation of the graphical representation of a diagram into an equivalent textual representation. We also present the axiomatic conventions that define the notation used, followed by definitions of the predefined atomic data types supported by the TIMEER model.

In section 3.2, we proceed to define the semantics of the basic data types supported by TIMEER, then define the semantic domains of the timestamps data types supported, followed by the semantics of the textual representation of the TIMEER model. Part of the running example is used to illustrate the main ideas behind the semantics.

3.1 Textual TimeER Diagrams

The translation from TIMEER diagrams to the equivalent textual representations is straightforward; given the TIMEER diagram in Figure 1, we will transform a part of this diagram in order to explain the transformation.

Before we present the full syntax of the textual representation of the TIMEER model, we describe the notation as well as conventions used in the abstract syntax of the textual representation and in the definition of the semantics.

Axiomatic Conventions

We let SET denote the class of sets, $FSET$ the class of finite sets, FUN the class of total functions, and REL the class of relations. The following inclusions hold $FSET \subseteq SET$ and $FUN \subseteq REL \subseteq SET$.

Next, assume that sets $S, \bar{S}_1, \dots, \bar{S}_n \in SET$ are given. We let $F(S)$ denote the restriction of the power set 2^S to finite sets, S^* denote the set of finite lists over S , S^+ the set of non-empty finite lists over S , and $S \times \bar{S}_1 \times \dots \times \bar{S}_n$ denote the Cartesian product over the sets $S, \bar{S}_1, \dots, \bar{S}_n$. The set of finite multisets over S is given by $M(S)$. A multiset can be considered a finite set S together with a counting function $occ : S \rightarrow N$, giving for each element the number of occurrences in the multiset. We let $S_1 \uplus S_2$ denote the disjoint union of sets, that is, the result of $S_1 \uplus S_2$ is $\{S_1, S_2\}$.

We write finite sets as $\{c_1, c_2, \dots, c_n\}$, lists as $\langle c_1, c_2, \dots, c_n \rangle$, elements of Cartesian products as (c_1, c_2, \dots, c_n) , and multisets as $\{\{c_1, c_2, \dots, c_n\}\}$. For a set

$\{c_1, c_2, \dots, c_n\}$, $i \neq j$ implies $c_i \neq c_j$. This is not necessarily true for multisets. Given multiset $\{\{c_1, c_2, \dots, c_n\}\}$ with $occ(c) = k$, there are k indices $i_1, \dots, i_k \in \{1, \dots, n\}$ with $c_{i_j} = c$ for $j \in \{1, \dots, k\}$. For any set, we use \perp to denote the undefined value of the set.

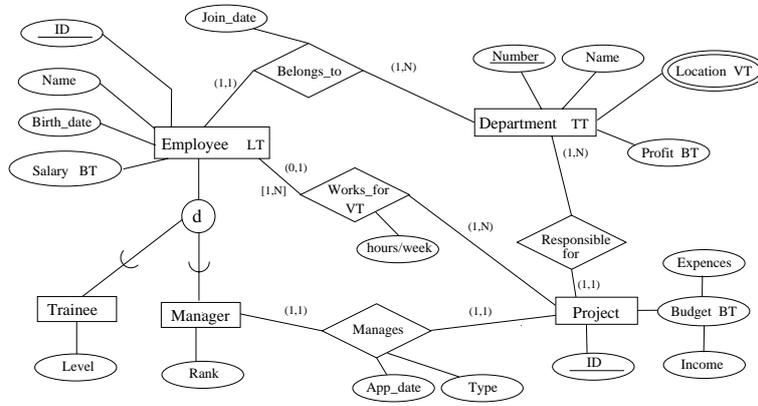


Figure 1: A TIMEER Diagram Describing a Company Database

Predefined Data Types

A data signature describes the predefined data types, operations, and predicates. We assume the data types *int*, *real*, and *string*; adding additional data types is straightforward. The data types *int*, *real*, and *string* and the operations and predicates on these have the usual semantics, and this interpretation is fixed, that is, defined once and for all. This definition follows the approach of Gogolla and Hohenstein and Khatri et al. [8, 14].

Let the syntax of a data signature DS be given as follows.

- the sets $DATA, OPNS, PRED \in FSET$
- a function $input \in FUN$ such that $input : OPNS \rightarrow DATA^*$
- a function $output \in FUN$ such that $output : OPNS \rightarrow DATA$
- a function $args \in FUN$ such that $args : PRED \rightarrow DATA^+$

If $\sigma \in OPNS$, $input(\sigma) = \langle d_1, \dots, d_n \rangle$, and $output(\sigma) = d$, this is denoted as $\sigma : \langle d_1, \dots, d_n \rangle \rightarrow d$. If $\pi \in PRED$ with $args(\pi) = \langle d_1, \dots, d_n \rangle$, this is denoted as $\pi : \langle d_1, \dots, d_n \rangle$

Example 3.1 The predefined data types and some operators and predicates working on the data types are given below.

$$DATA \supseteq \{ int, real, string \}$$

$$OPNS \supseteq \left\{ \begin{array}{lll} +_i, -_i, *_i : & int \times int & \rightarrow int, \\ +_r, -_r, *_r : & real \times real & \rightarrow real, \\ /_i : & int \times int & \rightarrow real, \\ /_r : & real \times real & \rightarrow real, \\ \uparrow_i : & int \times int & \rightarrow int, \\ \uparrow_r : & real \times int & \rightarrow real, \\ square_i : & int & \rightarrow int, \\ square_r : & real & \rightarrow real, \\ abs_i : & int & \rightarrow int, \\ abs_r : & real & \rightarrow real, \\ trc, rnd : & real & \rightarrow int, \\ cat : & string \times string & \rightarrow string \end{array} \right\}$$

$$PRED \supseteq \left\{ \begin{array}{l} <_i, >_i, \leq_i, \geq_i, \neq_i : int \times int, \\ <_r, >_r, \leq_r, \geq_r, \neq_r : real \times real, \\ <_s, >_s, \leq_s, \geq_s, \neq_s : string \times string \end{array} \right\}$$

Example 3.2 As a precursor to giving the textual representation of TIMEER diagrams, we transform the entity types Employee and Department, the relationship type Belongs_to, and the constraints related to these three modeling constructs into their textual representations.

For the entity type Employee, it is specified that both the lifespan and the transaction time of the instances must be captured. In the diagram in Figure 1, the data types of the timestamps are implicit; in the textual representation they are specified explicitly. The data type is temporal elements, and the granularity of the timestamps is hour for both temporal aspects to be captured. This results in the textual description below. Words in boldface are keywords.

Entity Type Employee with (*LS*, temporal element, hour), (*TT*, temporal element, hour)

We now have to add the attributes of the entity type Employee. It has the attributes ID, Name, Birth_date, and Salary. The only attribute where the temporal aspect is captured is Salary, and the time dimensions captured are valid time and transaction time. For all attributes, we have to specify the data type of the attribute values. For the temporal attributes, as for temporal entity types, the data type and the granularity of the timestamps capturing the temporal aspects are implicit in the diagrams, but have to be specified explicitly in the textual representation of the temporal attributes. The attributes of the Employee entity type are given next.

Attribute ID is of type int;
Attribute Name is of type string;
Attribute Birth_date is of type string;
Attribute Sal is of type real with
 (*VT*, temporal element, day), (*TT*, temporal element, day);

The translation of the other modeling constructs follow the same procedure and the textual representations of entity type Department and relationship type Belongs_to are as follows.

Entity Type Department with
 (*TT*, temporal element, day) **has**
Attribute Number is of type int;
Attribute Name is of type string;
Attribute Location is Multivalued of type string with
 (*VT*, temporal element, day);
Attribute Profit is of type real with
 (*VT*, temporal element, month),
 (*TT*, temporal element, month);

Relationship Type Belongs_to has
Attribute Join_date is of type string;
involves Employee; Department;

We now add key constraints to the entity types and snapshot participation constrains to the relationship type. ID is the key of Employee and Number is the key of Department; the snapshot participation constraint on Employee is (1,1), and the snapshot participation constraint on Department is (1,N). This gives us the following textual representation of the constraints.

ID is key of Employee;
Number is key of Department;
participation of Employee in Belongs_to is (1,1);
participation of Department in Belongs_to is (1,N);

The full syntax of the textual representation of the TIMEER model is given in Appendix A.

3.2 Semantics of TimeER

We are now able to define the semantics of the TIMEER model. First, we define the semantics of the predefined data types and define the model of time used in the semantics. Next, we explain the ideas behind the semantics, followed by the full semantics of the TIMEER model.

The semantics of a data signature DS is given by three functions.

A function $\mathcal{D}[\text{DATA}] \in FUN$ such that $\mathcal{D}[\text{DATA}] : \text{DATA} \rightarrow SET$ and $\perp \in \mathcal{D}[\text{DATA}](d)$ for every $d \in \text{DATA}$. The membership of $\perp \in \mathcal{D}[\text{DATA}](d)$ is required because it is necessary to have an undefined value as a result of an incorrect application of a function.

A function $\mathcal{D}[\text{OPNS}] \in FUN$ such that $\mathcal{D}[\text{OPNS}] : \text{OPNS} \rightarrow FUN$ and $\sigma : d_1 \times \dots \times d_n \rightarrow d$ implies $\mathcal{D}[\text{OPNS}](\sigma) : \mathcal{D}[\text{DATA}](d_1) \times \dots \times \mathcal{D}[\text{DATA}](d_n) \rightarrow \mathcal{D}[\text{DATA}](d)$ for every $d \in \text{DATA}$.

A function $\mathcal{D}[\text{PRED}] \in FUN$ such that $\mathcal{D}[\text{PRED}] : \text{PRED} \rightarrow REL$ and $\pi : d_1 \times \dots \times d_n$ implies $\mathcal{D}[\text{PRED}](\pi) \subseteq \mathcal{D}[\text{DATA}](d_1) \times \dots \times \mathcal{D}[\text{DATA}](d_n)$ for every $d \in \text{DATA}$.

Example 3.3 The semantics of the predefined data types in Example 3.1 and some of the associated operations are defined as follows.

$$\begin{aligned} \mathcal{D}[\text{DATA}](int) &= \mathbb{Z} \cup \{\perp\} \\ \mathcal{D}[\text{DATA}](real) &= \mathbb{R} \cup \{\perp\} \\ \mathcal{D}[\text{DATA}](string) &= \mathbb{A}^* \cup \{\perp\} \\ \mathcal{D}[\text{OPNS}](+) &: \mathcal{D}[\text{DATA}](int) \times \mathcal{D}[\text{DATA}](int) \rightarrow \mathcal{D}[\text{DATA}](int) \\ &= \begin{cases} (i_1, i_2) \rightarrow i_1 + i_2 & \text{if } i_1, i_2 \in \mathbb{Z} \\ \perp & \text{otherwise} \end{cases} \\ \mathcal{D}[\text{OPNS}](square_r) &: \mathcal{D}[\text{DATA}](real) \rightarrow \mathcal{D}[\text{DATA}](real) \\ &= \begin{cases} r \rightarrow r * r & \text{if } r \in \mathbb{R} \\ \perp & \text{otherwise} \end{cases} \end{aligned}$$

The Time Model

We assume that the real time line is bounded in both ends, so that time begins at the ‘‘Big Bang’’ and ends at the ‘‘Big Crunch.’’ A point t on the real time line is called an instant. The real time line is represented in the database by a so-called baseline clock [3]. In accord with the general consensus in the database community that a discrete model of time is adequate, the base-line clock, and thus our time domains, is discrete. Our time domains are then ordered, finite sets of elements isomorphic to finite subsets of the natural numbers. The elements are termed chronons. This may be seen as dividing the real time line into indivisible equal-size segments (the chronons). Real-world time instants are represented in the model by the chronons during which they occur. We will use c , possibly indexed, to denote chronons. The size of a chronon, called the granularity of the chronon, can be specified explicitly.

We introduce a domain for each combination of the temporal aspects and granularities supported. These domains are given by D_{dim}^g . The different valid-time domains are given as $D_{VT}^g = \{c_1^v, c_2^v, \dots, c_n^v\}$. The domain of all valid times is given as $\mathcal{D}_{VT} = \cup_g D_{VT}^g$. The transaction-time domains are given as $\mathcal{D}_{TT}^g = \{c_1^t, c_2^t, \dots, c_{now}^t\} \cup \{UC\}$ where UC (‘‘until changed’’) is a special transaction-time marker. The domain of all transaction times is then $\mathcal{D}_{TT} = \cup_g \mathcal{D}_{TT}^g$. The different lifespan domains are given as

$\mathcal{D}_{LS}^g = \{c_1^l, c_2^l, \dots, c_{now}^l\}$, and the domain of all lifespan times is given as $\mathcal{D}_{LS} = \cup_g \mathcal{D}_{LS}^g$. Some chronons are expected to be in the future and some are expected to be in the past. The chronon c_{now} denotes the chronon representing the current time.

A time interval is defined as the time between two instants, a starting instant and a terminating instant. A time interval is thus represented by the sequence of consecutive chronons where each chronon represents the instants that occurred during the chronon. We may represent a sequence of chronons by the starting and the ending chronon. We define intervals $[c_i, c_j]^g$ where c_i is the starting chronon, c_j is the terminating chronon, and the size of the chronon is g . We let $[c_i, c_j]_{vt}^g, [c_i, c_j]_{tt}^g, [c_i, c_j]_{ls}^g$ denote intervals over the valid-time, transaction-time, and lifespan domains, respectively.

We also define temporal elements over time domains. A temporal element is a union of intervals and is represented by $I^g = [c_i, c_j]^g \cup \dots \cup [c_l, c_k]^g$. Since our time domains are discrete and finite, we can define a temporal element as an element of the set $2^{D_{dim}^g}$. We let $I_{vt}^g, I_{tt}^g, I_{ls}^g$ denote temporal elements over the valid-time, transaction-time, and the lifespan domains, respectively.

Semantics of Example 3.2

In order to better understand the ideas behind the semantics of TIMEER, we will explain in detail the semantics of the entity type Employee, the relationship type Belongs_to, the key constraint on Employee, and the snapshot participation constraint of Employee in Belongs_to.

An entity type in a TIMEER diagram defines an entity set. The attributes of an entity characterize the entity, and each attribute of an entity has a value domain. The association between a set of attributes $X = \{A_1, A_2, \dots, A_n\}$ and the set of value domains D is given by a function $dom : X \rightarrow D$. An entity together with its attributes can be regarded as a tuple. A tuple t over a set of attributes X is actually a function that associates each attribute $A_i \in X$ with a value from the value domain $dom(A_i)$. For attribute A , we denote this value $t[A]$. The TIMEER model uses surrogates to identify the entities, and therefore extend the entity type with a surrogate attribute. We use surrogates to identify entities due to the fact that user defined identifiers can be specified as time-varying.

The semantics of the entity type Employee is therefore a set of functions (tuples), termed *Employee*. The domain of each function t is the set of attribute names connected to the entity type and the surrogate attribute, s . The value domain of the attributes connected to the entity type Employee is determined by the semantics of the attribute declarations, while the value domain of the surrogate attribute is the set $D_S^{Employee}$ of surrogate values assigned to *Employee*. The mathematical description of the above is presented next.

$$\begin{aligned} \mathcal{E}[\text{Entity Type Employee} \dots] &= \{t \mid t \in FUN \wedge \\ dom(t) &= \{s, ID, Name, Birth_date, Sal\} \wedge t[s] \in D_S^{Employee} \wedge \\ t[ID] &\in \mathcal{A}[\text{Attribute ID is } A'_D] \wedge \\ t[Name] &\in \mathcal{A}[\text{Attribute Name is } A'_D] \wedge \\ t[Birth_date] &\in \mathcal{A}[\text{Attribute Birth_date is } A'_D] \wedge \\ t[Sal] &\in \mathcal{A}[\text{Attribute Sal is } A'_D] \wedge \\ \forall t_i, t_j, i \neq j &\Rightarrow t_i[s] \neq t_j[s] \} \end{aligned}$$

In the above description, we have not yet determined the value domains of the attributes. The attribute *ID* is specified as non-temporal with data type *int*. This means that we do not want to capture the changes of this attribute over time. The semantics is therefore modeled as a constant belonging to the set

of integers, i.e., the value domain of this attribute is the set of integers, including the undefined value.

$\mathcal{A}[\text{Attribute } ID \text{ is of type } int] = \mathcal{D}[\text{DATA}](int) = \mathbb{Z} \cup \{\perp\}$

The *Birth_date* of an employee never changes, so this attribute is also described as non-temporal, but here we use the data type *string*. The value is therefore modeled as a constant sentence defined over some alphabet, i.e., the value domain is some alphabet, again including the undefined value.

$\mathcal{A}[\text{Attribute } Birth_date \text{ is of type } string] = \mathcal{D}[\text{DATA}](string) = \mathbb{A}^* \cup \{\perp\}$

We will not explain in detail the semantics of the attribute *Name*, but proceed to the attribute *Sal*. This attribute is a temporal attribute with data type *real*, that is, we want to record how the values of this attribute change over time. This means that the value domain of this attribute must be a function from some time domain to a value domain. The temporal aspects to be captured for this attribute are valid time and transaction time.

$\mathcal{A}[\text{Attribute } Sal \text{ is of type } real \text{ with } (VT, \text{temporal element, day}), (TT, \text{temporal element, day})] = \mathcal{T}[(VT, \text{temporal element, day})] \times \mathcal{T}[(TT, \text{temporal element, day})] \rightarrow \mathcal{D}[\text{DATA}](real) = D_{VT}^{day} \times D_{TT}^{day} \rightarrow \mathbb{R}^* \cup \{\perp\}$

The resulting, full semantics of the entity type *Employee* is presented next.

$\mathcal{E}[\text{Entity Type } Employee \dots] = \{t \mid t \in FUN \wedge dom(t) = \{s, ID, Name, Birth_date, Sal\} \wedge t[s] \in D_S^{Employee} \wedge t[ID] \in \mathbb{Z} \cup \{\perp\} \wedge t[Name] \in \mathbb{A}^* \cup \{\perp\} \wedge t[Birth_date] \in \mathbb{A}^* \cup \{\perp\} \wedge t[Sal] \in D_{VT}^{day} \times D_{TT}^{day} \rightarrow \mathbb{R} \cup \{\perp\}\}$

The key constraint of an entity set is a set of predicates the entity set has to satisfy. In the textual representation, all constraints are separate constructs, so we first have to check if the entity type mentioned in the constraint construct exists at all by calling the auxiliary predicate $inSch(Employee, Sc_D)$. The predicate is explained in detail on page 7. We also have to check that the set of attributes mentioned in the constraint really are attributes of the entity type. Next, we define the predicate that ensures that the values of the key attributes are unique for the entity set. The key constraint on *Employee* is described next.

$\mathcal{C}[ID \text{ is key of } Employee] = inSch(Employee, Sc_D) \wedge ID \in attOf(\text{Entity Type } Employee \text{ has } A_D) \wedge \forall t_i, t_j \in \mathcal{E}[\text{Entity Type } Employee \text{ has } A_D] (t_i[ID] = t_j[ID] \Rightarrow t_i[s] = t_j[s])$

A relationship type in a TIMEER diagram defines a relationship set. Its semantics is therefore a set of relationships. The relationship type *Belongs_to* describes relationships among entities from the entity types *Employee* and *Department*. We use the surrogates of the participating entities to identify which entities participate in which relationship(s). As for entities, we can regard each relationship in a set of relationships as an element of a Cartesian product over a set of attributes. The attributes of a relationship are the attributes of the relationship type and a surrogate attribute for each participating entity type. To identify the participating entity types, we use the auxiliary function $parOf(I_S)$ that takes an involvement specification as input and returns a set (or if the relationship type involves the same entity type more than once, a multiset) of entity type names. The semantics of the relationship type *Belongs_to* is given next.

$\mathcal{R}[\text{Relationship Type } Belongs_to \text{ has } A_D \text{ involves } I_S] = \{t \mid t \in FUN \wedge dom(t) = \bigcup_{E_i \in parOf(I_S)} s_{E_i} \cup \{Join_date\} \wedge \bigwedge_{E_i \in parOf(I_S)} t[s_{E_i}] \in \mathcal{I}[E_i] \wedge t[Join_date] \in \mathcal{A}[\text{Attribute } Join_date \text{ is } A'_D]\} = \{t \mid t \in FUN \wedge dom(t) = \{s_{Employee}, s_{Department}, Join_date\} \wedge t[s_{Employee}] \in D_S^{Employee} \wedge t[s_{Department}] \in D_S^{Department} \wedge t[Join_date] \in \mathbb{A}^* \cup \{\perp\}\}$

Snapshot participation constraints are like key constraints and define predicates that relationship sets have to satisfy. Again, since the participation constraints are separate constructs in the textual representation, we have to ensure that the entity type and the relationship type mentioned in each constraint exist. To count the number of relationships, an entity participate in, we use the auxiliary function $cnt(e, E, \mathcal{R}[R_D])$ that takes an entity, an entity type, and a relationship set as input and returns the number of relations in the relationship set, the entity e participates in.

$\mathcal{C}[\text{Snapshot participation of } Employee \text{ in } Belongs_to \text{ is } (1, 1)] = inSch(Employee, Sc_D) \wedge inSch(Belongs_to, Sc_D) \wedge Employee \in parOf(\text{Relationship Type } Belongs_to \text{ has } A_D \text{ involves } I_S) \wedge \forall e_j \in D_S^{Employee} (min \leq cnt(e_j, Employee, \mathcal{R}[\text{Relationship Type } Belongs_to \text{ has } A_D \text{ involves } I_S]) \leq max)$

The full semantics of the TIMEER model follow. First, we define the semantic domains. Second, we define the auxiliary functions to be used in the semantic functions. Finally, we define the semantic functions.

Semantic Domains

D_S – The set of surrogates
 $D_S^E \subseteq D_S$ – Surrogates assigned to $E \in [E_TYPE]$
 $D_S^R \subseteq D_S$ – Surrogates assigned to $R \in [R_TYPE]$
 $D_{LS} = (\bigcup_g D_{LS}^g) \cup \{\perp\}$ – The lifespan domains
 $D_{VT} = (\bigcup_g D_{VT}^g) \cup \{\perp\}$ – The valid time domains
 $D_{TT} = (\bigcup_g D_{TT}^g) \cup \{\perp\}$ – The transaction time domains
 $\mathcal{D}[\text{DATA}]$ – The set of basic domains

Auxiliary Functions

The function $attOf$ takes as input an entity type declarations and returns the list of attributes names of the entity type.

$attOf(\text{Entity Type } E \text{ has } A_D) = attOf(\text{Entity Type } E \text{ with } T_S \text{ has } A_D) = attOf(\text{Subclass } E_1 \text{ of } E_2 \text{ has } A_D) = attOf(\text{Subclass } E_1 \text{ of } E_2 \text{ with } T_S \text{ has } A_D) = attOf(A_D)$

$attOf(A_{D_1}; A_{D_2}) = attOf(A_{D_1}) \cup attOf(A_{D_2})$
 $attOf(\text{Attribute } A \text{ is } A'_D) = A$

The function $parAtt$ takes the name of an entity type as argument and returns the names of attributes of the entity type and its ancestor(s), if the entity type is declared as a subclass.

$parAtt(E) = \begin{cases} attOf(\text{Entity Type } E \dots) & \text{if } \text{Entity Type } E \dots \in E_D \\ attOf(\text{Subclass } E_1 \text{ of } E_2 \dots) \cup parAtt(E_2) & \text{if } \text{Subclass } E_1 \text{ of } E_2 \dots \in E_D \\ \perp & \text{otherwise} \end{cases}$

The function $parOf$ takes a relationship type declaration as argument and returns the entity types that participate in the relationship type.

$parOf(\text{Relationship Type } R \text{ has } A_D \text{ involves } I_S) =$
 $parOf(\text{Relationship Type } R \text{ with } T_S \text{ has } A_D \text{ involves } I_S)$
 $= parOf(I_S)$

$parOf(I_{S_1}; I_{S_2}) = parOf(I_{S_1}) \cup parOf(I_{S_2})$
 $parOf(E) = \{\{E\}\}$
 $parOf(E(\text{identifies})) = \{\{E\}\}$

The function *tempSpec* takes either a relationship type declarations or an entity type declaration as argument. If the declaration is non-temporal, it returns the empty set; and if the declaration is temporal, the specification of the required temporal support is returned.

$tempSpec(R) =$
 $\begin{cases} T_S & \text{if Relationship Type } R \text{ with } T_S \text{ has } A_D \\ & \text{involves } I_S \in R_D \\ \emptyset & \text{if Relationship Type } R \text{ has } A_D \text{ involves } I_S \in R_D \\ \perp & \text{otherwise} \end{cases}$
 $tempSpec(E) =$
 $\begin{cases} T_S & \text{if Entity Type } E \text{ with } T_S \text{ has } A_D \in E_D \\ \emptyset & \text{if Entity Type } E \text{ has } A_D \in E_D \\ T_S \times tempSpec(E_2) & \text{if Subclass } E_1 \text{ of } E_2 \text{ with } T_S \text{ has } A_D \in E_D \\ \emptyset \times tempSpec(E_2) & \text{if Subclass } E_1 \text{ of } E_2 \text{ has } A_D \in E_D \\ \perp & \text{otherwise} \end{cases}$

The function *ownerOf* takes as arguments the name of a weak entity type and an identifying relationship type declaration and returns the list of entity type names of the owners of the weak entity type.

$ownerOf(E, \text{Relationship Type } R \text{ with } T_S$
 $\text{has } A_D \text{ involves } I_S) =$
 $ownerOf(E, \text{Relationship Type } R \text{ has } A_D \text{ involves } I_S) =$
 $\begin{cases} parOf(I_S) - E & \text{if } E(\text{identifies}) \in I_S \\ \emptyset & \text{otherwise} \end{cases}$

The predicate *inSch* takes as arguments either an entity type name or a relationship type name, as well as a schema declaration. The predicate returns *true* if the entity type or the relationship type is declared in the schema and is *false* otherwise.

$inSch(E, Sc_D) = inSch(E, E_D; R_D; IC_D) = inSch(E, E_D) =$
 $\begin{cases} true & \text{if Entity Type } E \text{ with } T_S \text{ has } A_D \in E_D \\ true & \text{if Entity Type } E \text{ has } A_D \in E_D \\ false & \text{otherwise} \end{cases}$
 $inSch(R, Sc_D) = inSch(R, E_D; R_D; IC_C) = inSch(R, R_D) =$
 $\begin{cases} true & \text{if Relationship Type } R \text{ with } T_S \text{ has } A_D \\ & \text{involves } I_S \in R_D \\ true & \text{if Relationship Type } R \text{ has } A_D \text{ involves } I_S \in R_D \\ false & \text{otherwise} \end{cases}$

The function *cnt* takes an entity, an entity type, and a relationship set as inputs and returns the number of relations in the relationship set, the entity *e* participates in.

$cnt(e, E, \{t_1, \dots, t_n\}) =$
 $\begin{cases} 0 & \text{if } n = 0 \\ cnt(e, E, \{t_1, \dots, t_{n-1}\}) & \text{if } n \geq 1 \wedge t_n[s_E] \neq e \\ cnt(e, E, \{t_1, \dots, t_{n-1}\}) + 1 & \text{if } n \geq 1 \wedge t_n[s_E] = e \end{cases}$

Semantic Functions

In what follows we describe the semantic functions. First we present the signature of the semantic functions and then the semantic function are defined.

$\mathcal{I} : E_TYPE \rightarrow D_S^E$
 $\mathcal{T} : T_SPEC \rightarrow D_{VT} \cup D_{TT} \cup D_{LS}$
 $\mathcal{A} : ATT \times DATA \times T_SPEC \rightarrow \mathcal{D}[DATA] \cup F(2^{\mathcal{D}[DATA]}) \cup$
 $(\mathcal{T}[[T_S] \rightarrow \mathcal{D}[DATA]]) \cup (\mathcal{T}[[T_S] \rightarrow F(2^{\mathcal{D}[DATA]})])$
 $\mathcal{E} : E_TYPE \times T_SPEC \times A_DECL \rightarrow D_S^E \times \mathcal{A}[A_D] \cup$
 $(D_S^E \times \mathcal{T}[[T_S]]) \times \mathcal{A}[A_D]$
 $\mathcal{R} : R_TYPE \times I_SPEC \times T_SPEC \times A_DECL \rightarrow$
 $(D_S^R \times \mathcal{I}[[I_S]] \times \mathcal{T}[[T_S]]) \times \mathcal{A}[A_D] \cup (\mathcal{I}[[I_S]] \times \mathcal{T}[[T_S]]) \times$
 $\mathcal{A}[A_D] \cup \mathcal{I}[[I_S]] \times \mathcal{A}[A_D]$
 $\mathcal{C} : IC_D \rightarrow PRED$
 $\mathcal{S} : Sc_D \rightarrow \mathcal{S}[Sc_D]$

The purpose of the semantic function \mathcal{I} is to determine the surrogate sets of the entity types that are involved in a specific relationship type.

$\mathcal{I}[[I_{S_1}; I_{S_2}]] = \mathcal{I}[[I_{S_1}]] \times \mathcal{I}[[I_{S_2}]]$

$\mathcal{I}[[E]] = \begin{cases} D_S^E & \text{if } E \in E_TYPE \\ \perp & \text{otherwise} \end{cases}$

The semantic function \mathcal{T} is defined in order to determine the time domains of the specified temporal support for a given entity type, relationship type, or attribute.

$\mathcal{T}[[\text{with } T_{S_1}; T_{S_2}]] = \mathcal{T}[[T_{S_1}]] \times \mathcal{T}[[T_{S_2}]]$

$\mathcal{T}[[\langle dim, instant, g \rangle]] = D_{dim}^g$

$\mathcal{T}[[\langle dim, temporal\ element, g \rangle]] = 2^{D_{dim}^g}$

The semantic function \mathcal{A} is a function that given a specific attribute specification as input it returns the value domain of the specified attribute.

$\mathcal{A}[[A_{D_1}; A_{D_2}]] = \mathcal{A}[[A_{D_1}]] \times \mathcal{A}[[A_{D_2}]]$

$\mathcal{A}[[\text{Attribute } A \text{ is } A'_D]] = \mathcal{A}[[A'_D]]$

$\mathcal{A}[[\text{of type } d]] = \mathcal{D}[DATA](d)$

$\mathcal{A}[[\text{of type } d \text{ with } T_S]] = \mathcal{T}[[T_S]] \rightarrow \mathcal{D}[DATA](d)$

$\mathcal{A}[[\text{composite}(A_D)]] = \mathcal{A}[[A_D]]$

$\mathcal{A}[[\text{composite}(A_D) \text{ with } T_S]] = \mathcal{T}[[T_S]] \rightarrow \mathcal{A}[[A_D]]$

$\mathcal{A}[[\text{Multivalued of type } d]] = F(2^{\mathcal{D}[DATA](d)})$

$\mathcal{A}[[\text{Multivalued of type } d \text{ with } T_S]] =$
 $\mathcal{T}[[T_S]] \rightarrow F(2^{\mathcal{D}[DATA](d)})$

The semantic function \mathcal{E} identifies the set of functions (tuples) that are defined through the definition of entity types. Each entity type defines a unique set of tuples and these sets are stored separately in the database. This can be observed by the fact the function \mathcal{E} applied to a composition of entity type definitions returns the disjoint union of the function applied to each of the components of the composition. The function \mathcal{E} is applicable to all types of entity types.

$\mathcal{E}[[E_{D_1}; E_{D_2}]] = \mathcal{E}[[E_{D_1}]] \uplus \mathcal{E}[[E_{D_2}]]$

$\mathcal{E}[[\text{Entity Type } E \text{ has } A_D]] = \{t \mid t \in FUN \wedge$
 $dom(t) = \{s, attOf(A_D)\} \wedge t[s] \in \mathcal{I}[[E]]$
 $\bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[[\text{Attribute } A_i \text{ is } A'_D]] \wedge$
 $\forall c^l \in \mathcal{D}_{LS} \forall c^t \in \mathcal{D}_{TT} (\forall t_i, t_j, i \neq j \Rightarrow t_i[s] \neq t_j[s])\}$

$\mathcal{E}[[\text{Entity Type } E \text{ with } T_S \text{ has } A_D]] = \{t \mid t \in FUN \wedge$
 $dom(t) = \{s, attOf(A_D)\} \wedge t[s] \in (\mathcal{T}[[T_S]] \rightarrow \mathcal{I}[[E]])$
 $\bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[[\text{Attribute } A_i \text{ is } A'_D]] \wedge$
 $\forall c^l \in \mathcal{D}_{LS} \forall c^t \in \mathcal{D}_{TT} (\forall t_i, t_j, i \neq j \Rightarrow t_i[s] \neq t_j[s])\}$

$\mathcal{E}[[\text{Weak Entity Type } E \text{ has } A_D]] = \{t \mid t \in FUN \wedge$
 $dom(t) = \{\bigcup_{E_i \in ownerOf(E, R_D)} s_{E_i}, attOf(A_D)\}$
 $\bigwedge_{E_i \in ownerOf(E, R_D)} t[s_{E_i}] \in \mathcal{I}[[E_i]]$
 $\bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[[\text{Attribute } A_i \text{ is } A'_D]]\}$

$\mathcal{E}[[\text{Weak Entity Type } E \text{ with } T_S \text{ has } A_D]] =$
 $\{t \mid t \in FUN \wedge dom(t) =$
 $\{\bigcup_{E_i \in ownerOf(E, R_D)} s_{E_i}, attOf(A_D)\}$
 $\bigwedge_{E_i \in ownerOf(E, R_D)} t[s_{E_i}] \in \mathcal{T}[[T_S]] \rightarrow \mathcal{I}[[E_i]]$
 $\bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[[\text{Attribute } A_i \text{ is } A'_D]]\}$

$$\begin{aligned} \mathcal{E}[\text{Subclass } E_1 \text{ of } E_2 \text{ has } A_D] &= \{t \mid t \in FUN \wedge \\ & \text{dom}(t) = \{s_{E_2}, \text{parAtt}(E_2), \text{attOf}(A_D)\} \wedge \\ & t[s_{E_2}] \in \mathcal{T}[\text{tempSpec}(E_2)] \rightarrow D_S^{E_2} \\ & \wedge_{A_i \in \text{parAtt}(E_2)} t[A_i] \in \mathcal{A}[\text{Attribute } A_i \text{ is } A'_D] \\ & \wedge_{A_i \in \text{attOf}(A_D)} t[A_i] \in \mathcal{A}[\text{Attribute } A_i \text{ is } A'_D]\} \\ \mathcal{E}[\text{Subclass } E_1 \text{ of } E_2 \text{ with } T_S \text{ has } A_D] &= \\ & \{t \mid t \in FUN \wedge \text{dom}(t) = \{s_{E_2}, \text{parAtt}(E_2), \text{attOf}(A_D)\} \\ & \wedge t[s_{E_2}] \in \mathcal{T}[\text{tempSpec}(E_2)] \times \mathcal{T}[T_S] \rightarrow D_S^{E_2} \\ & \wedge_{A_i \in \text{parAtt}(E_2)} t[A_i] \in \mathcal{A}[\text{Attribute } A_i \text{ is } A'_D] \\ & \wedge_{A_i \in \text{attOf}(A_D)} t[A_i] \in \mathcal{A}[\text{Attribute } A_i \text{ is } A'_D]\} \end{aligned}$$

The semantic function \mathcal{R} determines the tuples defined through the definition of relationship types. As for entity types, relationship types defines a unique set of tuples that are stored separately in the database. The function is applicable not applicable to super-/sub class relationships as these relationships are being handled though the application of the semantic function \mathcal{E} on subclass entity types.

$$\begin{aligned} \mathcal{R}[\mathcal{R}_{D_1}; \mathcal{R}_{D_2}] &= [\mathcal{R}_{D_1}] \uplus [\mathcal{R}_{D_2}] \\ \mathcal{R}[\text{Relationship Type } R \text{ has } A_D \text{ involves } I_S] &= \\ & \{t \mid t \in FUN \wedge \text{dom}(t) = \\ & \{\bigcup_{E_i \in \text{parOf}(I_S)} s_{E_i}, \text{attOf}(A_D)\} \\ & \wedge_{E_i \in \text{parOf}(I_S)} t[s_{E_i}] \in \mathcal{I}[E_i] \\ & \wedge_{A_i \in \text{attOf}(A_D)} t[A_i] \in \mathcal{A}[\text{Attribute } A_i \text{ is } A'_D]\} \\ \mathcal{R}[\text{Relationship Type } R \text{ with } T_S \text{ has } A_D \text{ involves } I_S] &= \\ & \left\{ \begin{array}{l} \{t \mid t \in FUN \wedge \text{dom}(t) = \\ \quad \{\bigcup_{E_i \in \text{parOf}(I_S)} s_{E_i}, \text{attOf}(A_D)\} \\ \quad \wedge_{E_i \in \text{parOf}(I_S)} t[s_{E_i}] \in \mathcal{T}[T_S] \rightarrow \mathcal{I}[E_i] \\ \quad \wedge_{A_i \in \text{attOf}(A_D)} t[A_i] \in \mathcal{A}[\text{Attribute } A_i \text{ is } A'_D] \\ \quad \text{if } \mathcal{T}[T_S] \notin D_{LS}^g \\ \{t \mid t \in FUN \wedge \text{dom}(t) = \\ \quad \{s_R, \bigcup_{E_i \in \text{parOf}(I_S)} s_{E_i}, \text{attOf}(A_D)\} \wedge \\ \quad t[s_R] \in \mathcal{T}[T_S] \rightarrow D_S^R \\ \quad \wedge_{E_i \in \text{parOf}(I_S)} t[s_{E_i}] \in \mathcal{T}[T_S] \rightarrow \mathcal{I}[E_i] \\ \quad \wedge_{A_i \in \text{attOf}(A_D)} t[A_i] \in \mathcal{A}[\text{Attribute } A_i \text{ is } A'_D] \\ \text{otherwise} \end{array} \right. \end{aligned}$$

The primary purpose of the semantic function \mathcal{C} is to ensure that the designed database is valid. The general design of the functions is first to ensure that the constructs mentioned in the constraint is in the schema definition and then to define the predicate that the objects involved in the constraint has to satisfy. The result of the application of the set of constraint functions is a set of predicates that the defined database has to satisfy.

$$\begin{aligned} \mathcal{C}[\mathcal{I}C_{D_1}; \mathcal{I}C_{D_2}] &= \mathcal{C}[\mathcal{I}C_{D_1}] \wedge \mathcal{C}[\mathcal{I}C_{D_2}] \\ \mathcal{C}[B \text{ is key of } E] &= \text{inSch}(E, Sc_D) \wedge \\ & ((B \subseteq \text{attOf}(\text{Entity Type } E \text{ has } A_D) \wedge \\ & \quad \forall t_i, t_j \in \mathcal{E}[\text{Entity Type } E \text{ has } A_D] \\ & \quad (t_i[B] = t_j[B] \Rightarrow t_i[s] = t_j[s])) \vee \\ & (B \subseteq \text{attOf}(\text{Entity Type } E \text{ with } T_S \text{ has } A_D) \wedge \\ & \quad \forall t_i, t_j \in \mathcal{E}[\text{Entity Type } E \text{ with } T_S \text{ has } A_D] \\ & \quad (\mathcal{T}[T_S] \rightarrow t_i[B] = \mathcal{T}[T_S] \rightarrow t_j[B] \Rightarrow \mathcal{T}[T_S] \rightarrow \\ & \quad t_i[s] = \mathcal{T}[T_S] \rightarrow t_j[s])))) \\ \mathcal{C}[B \text{ is partial key of } E] &= \text{inSch}(E, Sc_D) \wedge \\ & ((B \subseteq \text{attOf}(\text{Weak Entity Type } E \text{ has } A_D) \wedge \\ & \quad \forall t_i, t_j \in \mathcal{E}[\text{Weak Entity Type } E \text{ has } A_D] \\ & \quad (\bigcup_{E_i \in \text{ownerOf}(E)} t_i[s_{E_i}] = \bigcup_{E_i \in \text{ownerOf}(E)} t_j[s_{E_i}] \wedge \\ & \quad t_i[B] = t_j[B] \Rightarrow t_i = t_j)) \vee \\ & (B \subseteq \text{attOf}(\text{Weak Entity Type } E \text{ with } T_S \text{ has } A_D) \wedge \\ & \quad \forall t_i, t_j \in \mathcal{E}[\text{Weak Entity Type } E \text{ with } T_S \text{ has } A_D] \\ & \quad (\mathcal{T}[T_S] \rightarrow \bigcup_{E_i \in \text{ownerOf}(E)} t_i[s_{E_i}] = \\ & \quad \mathcal{T}[T_S] \rightarrow \bigcup_{E_i \in \text{ownerOf}(E)} t_j[s_{E_i}] \wedge \\ & \quad \mathcal{T}[T_S] \rightarrow t_i[B] = \mathcal{T}[T_S] \rightarrow t_j[B] \Rightarrow \\ & \quad \mathcal{T}[T_S] \rightarrow t_i = \mathcal{T}[T_S] \rightarrow t_j))) \end{aligned}$$

$$\begin{aligned} \mathcal{C}[\text{Participation of } I_S \text{ with respect to } E \text{ is } dis, tot] &= \\ & \text{inSch}(E) \wedge_{E_i \in I_S} \text{inSch}(E_i) \wedge \forall c^l \in \mathcal{D}_{LS} \forall c^t \in \mathcal{D}_{TT} \\ & (\forall t \in \bigcup_{E_i \in I_S} \mathcal{E}[\text{Subclass } E_i \text{ of } E \dots] \exists t' \in \mathcal{E}[\dots E \dots] \\ & \quad (t[s_{E_i}] = t'[s_{E_i}]) \wedge \\ & \quad \forall t \in \mathcal{E}[\dots E \dots] \exists t' \in \\ & \quad \mathcal{E}[\text{Subclass } E_i \text{ of } E \dots] (t[s_E] = t'[s_{E_i}]) \wedge \\ & \quad \forall E_i, E_j \in I_S \\ & \quad \nexists t_1 \in \mathcal{E}[\text{Subclass } E_i \text{ of } E \dots] t_2 \in \\ & \quad \mathcal{E}[\text{Subclass } E_j \text{ of } E \dots] (i \neq j \wedge t_1[s_{E_i}] = t_2[s_{E_j}])) \\ \mathcal{C}[\text{Participation of } I_S \text{ with respect to } E \text{ is } dis, par] &= \\ & \text{inSch}(E) \wedge_{E_i \in I_S} \text{inSch}(E_i) \wedge \forall c^l \in \mathcal{D}_{LS} \forall c^t \in \mathcal{D}_{TT} \\ & (\forall t \in \bigcup_{E_i \in I_S} \mathcal{E}[\text{Subclass } E_i \text{ of } E \dots] \exists t' \in \mathcal{E}[\dots E \dots] \\ & \quad (t[s_{E_i}] = t'[s_{E_i}]) \wedge \\ & \quad \forall E_i, E_j \in I_S \nexists t_1 \in \mathcal{E}[\text{Subclass } E_i \text{ of } E \dots] \\ & \quad t_2 \in \mathcal{E}[\text{Subclass } E_j \text{ of } E \dots] (i \neq j \wedge t_1[s_{E_i}] = t_2[s_{E_j}])) \\ \mathcal{C}[\text{Participation of } I_S \text{ with respect to } E \text{ is } over, tot] &= \\ & \text{inSch}(E) \wedge_{E_i \in I_S} \text{inSch}(E_i) \wedge \forall c^l \in \mathcal{D}_{LS} \forall c^t \in \mathcal{D}_{TT} \\ & (\forall t \in \bigcup_{E_i \in I_S} \mathcal{E}[\text{Subclass } E_i \text{ of } E \dots] \exists t' \in \mathcal{E}[\dots E \dots] \\ & \quad (t[s_{E_i}] = t'[s_{E_i}]) \wedge \\ & \quad \forall t \in \mathcal{E}[\dots E \dots] \exists t' \in \mathcal{E}[\text{Subclass } E_i \text{ of } E \dots] \\ & \quad (t[s_E] = t'[s_{E_i}])) \\ \mathcal{C}[\text{Participation of } I_S \text{ with respect to } E \text{ is } over, par] &= \\ & \text{inSch}(E) \wedge_{E_i \in I_S} \text{inSch}(E_i) \wedge \forall c^l \in \mathcal{D}_{LS} \forall c^t \in \mathcal{D}_{TT} \\ & (\forall t \in \bigcup_{E_i \in I_S} \mathcal{E}[\text{Subclass } E_i \text{ of } E \dots] \exists t' \in \mathcal{E}[\dots E \dots] \\ & \quad (t[s_{E_i}] = t'[s_{E_i}])) \\ \mathcal{C}[\text{Lifespan participation of } E \text{ in } R \text{ is } [min, max]] &= \\ & \text{inSch}(E, Sc_D) \wedge \text{inSch}(R, Sc_D) \wedge \\ & E \in (\text{parOf}(\text{Relationship Type } R \text{ with } T_S \\ & \quad \text{has } A_D \text{ involves } I_S) \wedge \\ & \quad \forall e_j \in D_S^E (min \leq \\ & \quad \text{cnt}(e_j, E, \mathcal{R}[\text{Relationship Type } R \text{ with } T_S \\ & \quad \text{has } A_D \text{ involves } I_S]) \leq max)) \\ \mathcal{C}[\text{Snapshot participation of } E \text{ in } R \text{ is } (min, max)] &= \\ & \text{inSch}(E, Sc_D) \wedge \text{inSch}(R, Sc_D) \wedge \\ & (E \in \text{parOf}(\text{Relationship Type } R \text{ with } T_S \\ & \quad \text{has } A_D \text{ involves } I_S) \vee \\ & E \in \text{parOf}(\text{Relationship Type } R \\ & \quad \text{has } A_D \text{ involves } I_S)) \wedge \\ & (\forall e_j \in D_S^E (min \leq \\ & \quad \text{cnt}(e_j, E, \mathcal{R}[\text{Relationship Type } R \\ & \quad \text{has } A_D \text{ involves } I_S]) \leq max)) \vee \\ & (\forall c \in \mathcal{T}[\text{tempSpec}(R)] \forall e_j \in D_S^E (min \leq \\ & \quad \text{cnt}(e_j, E, \mathcal{R}[\text{Relationship Type } R \\ & \quad \text{with } T_S \text{ has } A_D \text{ involves } I_S]) \leq max)) \end{aligned}$$

The last of the semantic functions \mathcal{S} defines exactly what a TIMEER diagram consists of. It also defines the different parts that define the objects of the underlying database and the predicates that ensures that the database is valid and consistent. The three different parts are: Entity type definitions, relationship type definitions, and integrity constraint definitions.

$$\begin{aligned} \mathcal{S}[Sc_D] &= \mathcal{S}[E_D; R_D; IC_D] \\ \mathcal{S}[E_D; R_D; IC_D] &= \mathcal{E}[E_D] \uplus \mathcal{R}[R_D] \uplus \mathcal{C}[IC_D] \end{aligned}$$

4 Conclusions and Research Directions

Temporal aspects are prevalent in most real-world database applications, but they are also difficult to capture elegantly using the ER model. As a result several temporally extended ER models have been presented by the database community. We have studied several of these models, and have found that the semantics of these models are either not defined or defined in terms of the relational model.

In this paper we present the formal semantics of a temporally extended ER model, the TIMEER model. The TIMEER model systematically extends the EER model [5] with new, enhanced modeling constructs with implicit temporal support. The semantics is based on denotational semantic and can therefore be useful for all database designers not only those that implement their database on a relational platform. The semantics can easily be applied to the other existing temporally extended models.

With respect to further work within this area one could extend the semantics presented in this paper to the *TIMEERplus* model [9] which extends the *TIMEER* model. The extension includes *graphical* notation for describing more temporal aspects of data, including the update and observation patterns for temporal attributes [13] and schema changes. The extent to which this is feasible is still unclear.

References

- [1] S. Bergamaschi and C. Sartori. Chrono: A Conceptual Design Framework for Temporal Entities. In T. W. Ling, S. Ram, and M-L. Lee, editors, *Conceptual Modeling - ER '98, 17th International Conference on Conceptual Modeling, Singapore, November 16-19, 1998, Proceedings*, volume 1507 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 1998.
- [2] P. P-S. Chen. The Entity-Relationship Model – Toward a Unified View of Data. *Transaction on Database Systems*, 1(1):9–36, March 1976.
- [3] C. E. Dyreson and R. T. Snodgrass. The Baseline Clock. In *The TSQL2 Temporal Query Language*, chapter 5, pages 77–96. Kluwer Academic Publishers, 1995.
- [4] R. Elmasri, I. El-Assal, and V. Kouramajian. Semantics of Temporal Data in an Extended ER Model. In *9th International Conference on the Entity-Relationship Approach*, pages 239–254, Lausanne, Switzerland, October 1990.
- [5] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, INC, 2. edition, 1994. ISBN 0-8053-1753-8.
- [6] R. Elmasri and G. T. J. Wu. A Temporal Model and Query Language for ER Databases. In *Proceedings of the Sixth International Conference on Data Engineering*, pages 76–83, 1990.
- [7] S. Ferg. Modeling the Time Dimension in an Entity-Relationship Diagram. In *4th International Conference on the Entity-Relationship Approach*, pages 280–286, Silver Spring, MD, 1985. Computer Society Press.
- [8] M. Gogolla and U. Hohenstein. Towards a Semantic View of an Extended Entity-Relationship Model. *ACM Transaction on Database Systems*, 16(3):369–416, September 1991.
- [9] H. Gregersen. *TimeERplus*: A Temporal EER Model Supporting Schema Changes. In *BNCOD*, volume 3567 of *Lecture Notes in Computer Science*, pages 41–59. Springer, 2005.
- [10] H. Gregersen and C. S. Jensen. Temporal Entity-Relationship Models—a Survey. *IEEE Transactions on Knowledge and Data Engineering*, 11(3):464–497, May 1999.
- [11] H. Gregersen and C. S. Jensen. Conceptual Modeling of Time-varying Information. In *Proceedings of International Conference on Computing, Communications and Control Technologies*, pages 248–255, August 2004.
- [12] C. S. Jensen and C. E. Dyreson[editors]. The Consensus Glossary of Temporal Database Concepts - February 1998 Version. In *Temporal Databases: Research and Practice*, volume 1399 of *Lecture Notes in Computer Science*, pages 367–405. Springer-Verlag, 1998.
- [13] C. S. Jensen and R. T. Snodgrass. Temporally Enhanced Database Design. In M. P. Papazoglou, S. Spaccapietra, and Z. Tari, editors, *Advances in Object-Oriented Data Modeling*, chapter 7, pages 163–193. MIT Press, 2000.
- [14] V. Khatri, S. Ram, and R. T. Snodgrass. Augmenting a Conceptual Model with Geospatiotemporal Annotations. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1324–1338, 2004.
- [15] M. R. Klopprogge. TERM: An Approach to Include the Time Dimension in the Entity-Relationship Model. In *Proceedings of the Second International Conference on the Entity Relationship Approach*, pages 477–512, Washington, DC, October 1981.
- [16] P. Kraft and J. O. Sørensen. Translation of a High-Level Temporal Model into Lower Level Models. In H. S.Kunii, S. Jajodia, and A. Sølvberg, editors, *Conceptual Modeling - ER 2001*, volume 2224 of *Lecture Notes in Computer Science*, pages 383–396. Springer, 2001.
- [17] V. S. Lai, J-P. KUILBOER, and J. L. Guynes. Temporal Databases: Model Design and Commercialization Prospects. *DATA BASE*, 25(3):6–18, 1994.
- [18] J. Y. Lee and R. Elmasri. An EER-Based Conceptual Model and Query Language for Time-Series Data. In T. W. Ling, S. Ram, and M-L. Lee, editors, *Conceptual Modeling - ER '98, 17th International Conference on Conceptual Modeling, Singapore, November 16-19, 1998, Proceedings*, volume 1507 of *Lecture Notes in Computer Science*, pages 21–34. Springer, 1998.
- [19] P. McBrien, A. H. Seltveit, and B. Wangler. An Entity-Relationship Model Extended to describe Historical information. In *International Conference on Information Systems and Management of Data*, pages 244–260, Bangalore, India, July 1992.
- [20] A. Narasimhalu. A Data Model for Object-Oriented Databases with Temporal Attributes and Relationships. Technical report, National University of Singapore, 1988.
- [21] R. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In *Proceedings of ACM-SIGMOD 1985 International Conference on Management of Data*, pages 236–246, Austin, TX, May 1985.
- [22] B. TAUZOVICh. Toward Temporal Extensions to the Entity-Relationship Model. In *The 10th International Conference on the Entity Relationship Approach*, pages 163–179, San Mateo, California, October 1991.
- [23] C. I. Theodoulidis, P. Loucopoulos, and B. Wangler. A Conceptual Modelling Formalism for Temporal Database Applications. *Information Systems*, 16(4):401–416, 1991.
- [24] E. Zimanyi, C. Parent, S. Spaccapietra, and A. Pirotte. TERC+: A Temporal Conceptual Model. In *Proc. Int. Symp. on Digital Media Information Base*, November 1997.

A The Textual Representation of TimeER

Meta variables

$Sc_D \in Schemadecls$
 $E_D \in Enttypedecls$
 $R_D \in Reltypedecls$
 $A_D \in Attributedecls$
 $IC_D \in ICdecls$
 $E \in E_TYPE$
 $R \in R_TYPE$
 $A \in ATT$
 $B \in 2^{ATT}$
 $T_S \in T_SPEC$
 $I_S \in I_SPEC$
 $d \in DATA$
 $max, min \in Integer\ constants$
 $dim \in \{LS, VT, TT\}$
 $ts \in \{instant, temporal\ element\}$
 $g \in \{sec, min, hour, day, week, month, year\}$
 $p_1, p_2 \in \{dis, over, tot, par\}$

In the above defined meta variables we have introduced some abbreviations. These appear in the last line where disjoint is abbreviated to dis, overlapping to over, total to tot, and partial to par.

Abstract Syntax

$Sc_D ::= E_D; R_D; IC_D$
 $E_D ::= E_{D_1}; E_{D_2}$
 | **Entity Type** E **has** A_D
 | **Entity Type** E **with** T_S **has** A_D
 | **Weak Entity Type** E **has** A_D
 | **Weak Entity Type** E **with** T_S **has** A_D
 | **Subclass** E_1 **of** E_2 **has** A_D
 | **Subclass** E_1 **of** E_2 **with** T_S **has** A_D
 $R_D ::= R_{D_1}; R_{D_2}$
 | **Relationship Type** R **has** A_D **involves** I_S
 | **Relationship Type** R **with** T_S **has** A_D
 | **involves** I_S
 $IC_D ::= IC_{D_1}; IC_{D_2}$
 | B **is primary key of** E
 | B **is partial key of** E
 | **Snapshot participation of** E **in** R **is**
 | (min, max)
 | **Lifespan participation of** E **in** R **is**
 | $[min, max]$
 | **Participation of** I_S **with respect to**
 | E **is** p_1, p_2
 $A_D ::= A_{D_1}; A_{D_2}$
 | **Attribute** A **is** A'_D
 $A'_D ::=$ **of type** d
 | **of type** d **with** T_S
 | **composite**(A_D)
 | **composite**(A_D) **with** T_S
 | **Multivalued of type** d
 | **Multivalued of type** d **with** T_S
 $T_S ::= T_{S_1}; T_{S_2}$
 | (dim, ts, g)
 $I_S ::= I_{S_1}; I_{S_2}$
 | E
 | E (**identifies**)
 $dim ::= LS \mid VT \mid TT$
 $ts ::= instant \mid temporal\ element$
 $g ::= sec \mid min \mid hour \mid day \mid week \mid month \mid year$
 $d ::= int \mid real \mid string$
 $p_1 ::= dis \mid over$
 $p_2 ::= tot \mid par$

A Conceptual Solution for Representing Time in Data Warehouse Dimensions*

Elzbieta Malinowski[†]

Esteban Zimányi

Department of Computer & Network Engineering
 Université Libre de Bruxelles,
 50 av. F.D.Roosevelt, 1050 Brussels, Belgium,
 E-mail: emalinow, ezimanyi@ulb.ac.be

Abstract

Data Warehouses (DWs) use an omnipresent time dimension for keeping track of changes in measure values. However, this dimension cannot be used to model changes in other dimensions. On the other hand, Temporal Databases (TDBs) have been successfully used for modelling time-varying information. Bringing together these two research areas, leading to Temporal Data Warehouses (TDWs), provides the necessary solutions for managing time-varying data in dimensions. In this paper, we introduce temporal extensions for the MultiDimER model, a conceptual multidimensional model. In our model we allow the inclusion of valid and transaction time, which are obtained from source systems, in addition to the data warehouse loading time. Our model allows a conceptual representation of time-varying levels, attributes, and hierarchies. For the latter, we discuss different cases depending on whether the changes in levels affect the relationships between them.

Keywords: Data warehouses, conceptual modelling, temporal data warehouse design, time-varying levels, time-varying hierarchies.

1 Introduction

Decision-making users increasingly rely on Data Warehouses (DWs) to access historical data for supporting the strategic decisions of organizations. A DW is “a collection of subject-oriented, integrated, non-volatile, and time-variant data to support management’s decisions” (Inmon 2002). Subject orientation means that the development of DWs is done according to the analytical necessities of managers on different levels of the decision-making process. Integration represents the complex effort to join data from different operational and external systems. Non-volatility ensures data durability and time-variation indicates the possibility to keep different values of the same information according to its changes in time. Therefore, the last two features indicate that DWs should allow changes to data without overwriting values of already existing data.

The structure of a DW is based on a multidimensional view of data usually represented at a logical level using a *star* or *snowflake* schema, consisting of fact and dimension tables (Figure 1).

*The work of E. Malinowski was funded by a scholarship of the Cooperation Department of the Université Libre de Bruxelles.

[†]Currently on leave from the Universidad de Costa Rica.
 Copyright ©2006, Australian Computer Society, Inc. This paper appeared at Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Australia. Conferences in Research and Practice in Information Technology, Vol. 53. Markus Stumptner, Sven Hartmann, and Yasushi Kiyoki, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

A *fact table* (e.g., Sales facts in Figure 1) represents the subject orientation and the focus of analysis, e.g., analysis of sales. It usually contains numeric data called *measures* (e.g., Quantity or Sales in Figure 1) representing analysis needs in quantified form. *Dimensions* (e.g., Product, Time, and Store in Figure 1) are used for exploring the measures from different analysis perspectives. They usually contain hierarchies, such as Product–Category–Department in Figure 1. Further, a dimension may also have descriptive attributes, e.g., Store number or Manager’s name in the Store dimension.

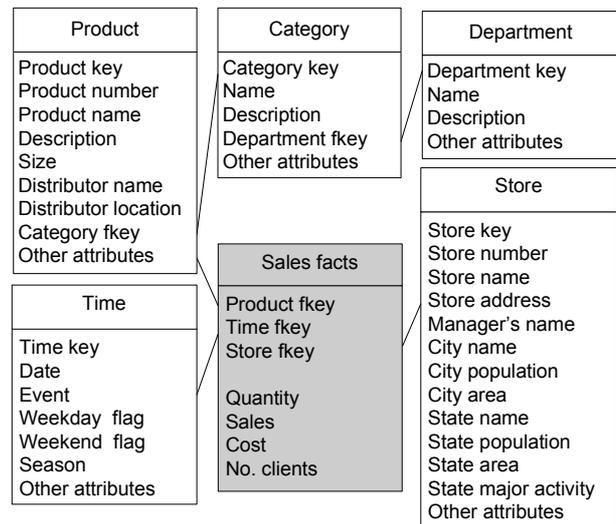


Figure 1: A snowflake schema for a Sales Data Warehouse.

On-Line Analytical Processing (OLAP) systems allow decision-making users to dynamically manipulate the data contained in a DW. OLAP systems use a structure called a *cube* which is also based on dimensions, measures, and hierarchies. Hierarchies allow both detailed and generalized view of data using the roll-up and drill-down operations. Further, the slice and dice operations allow to select a portion of the data based on specified values in one or several dimensions.

Current DW and OLAP models include an omnipresent time dimension that, as the other dimensions, is used for grouping purposes (the roll-up operation) or in a predicate role (the slice and dice operations). For example, if the measure Sales in Figure 1 is represented on a monthly basis, selecting some quarter in the Time dimension will aggregate all monthly measures corresponding to that quarter.

Nevertheless, even though the time dimension additionally serves as a time-varying indicator for measures, e.g., total sales in March 2005, it cannot be used for representing the time when changes

in other dimensions have occurred (Eder, Koncilia & Morzy 2002). Therefore, usual multidimensional models are not symmetric in the way of representing changes for measures and dimensions: while they allow to track changes in measure values, they are not able to represent changes in dimension data and the time when these changes have occurred, e.g., when a product has changed its ingredients. Consequently, the features of “time-variant” and “non-volatility” only apply for measures leaving to applications the representation of changes occurring in dimensions.

To represent these changes, several implementation solutions were proposed for relational databases for the so-called *slowly-changing dimensions* (Kimball, Ross & Merz 2002). However, some of them do not preserve the entire history of data. Another solution allows to reflect changes to data but requires significant programming effort for managing and querying time-varying dimension data. Further, these solutions do not consider research related to managing time-varying information in temporal databases.

Temporal databases (TDBs) have been investigated over the last decades, e.g., (Elmasri & Wuu 1990, Snodgrass 1995). They provide structures and mechanisms for representing and managing information that vary over time. Two different temporal types¹ are usually considered: *valid time* (VT) and *transaction time* (TT) that allow to represent, respectively, when the data is true in the modelled reality and when it is current in the database. If both temporal types are used, they define *bitemporal time* (BT). Further, in some applications, changes in time can be defined for an object as a whole, i.e., recording *lifespan* (LS) or *existence time* of a database object.

These temporal types are used for representing either *events*, i.e., something that happens at a particular time point, or *states*, i.e., something that has extent over time. For the former an *instant* is used, i.e., a time point on an underlying time axis. A state is represented by an *interval* or *period* indicating the time between two instants using either a non-anchored (e.g., two weeks) or an anchored length of time (e.g., [02/11/2004,05/01/2005]), respectively. Sets of instants and sets of intervals can also be used for representing events and states.

Temporal Data Warehouses (TDWs) join the research achievements of Temporal Databases and Data Warehouses in order to manage time-varying multidimensional data. TDWs raise many issues including consistent aggregation in presence of time-varying data, temporal queries of multidimensional data, storage methods, temporal view materialization, etc. Nevertheless, very little attention from the research community has been drawn to conceptual modelling for TDWs and to the analysis of which temporal support should be included in TDWs considering that TDBs and DWs are semantically different:

- DW data is integrated from existing source systems. TDB data is inserted by users since it represents operational or transactional databases. Therefore, different temporal support may exist according to the types of source systems that integrate data into a TDW.
- DWs support the decision-making process, while TDBs reflect data changes in the reality (VT) and in the database content (TT). To expand the analysis spectrum for decision-making users different temporal types should also be considered in TDWs.

¹Usually called time dimensions; however, we use the term “dimension” in the multidimensional context.

- DW data is neither modified nor deleted². In contrast, in TDBs users change data directly usually recording the time of these changes as TT. Thus, the TT generated in a TDW plays a different role from the TT used in a TDB.
- DWs are designed according to analysis needs of decision-making users mostly based on a multidimensional view of data with clearly distinguished measures and dimensions. The last two play different roles: measures are aggregated while dimensions are used to explore measures according to different criteria. On the other hand, TDB design is concerned with transactional or operational applications where all data is handled in a similar manner. Therefore, the analysis of temporal support for multidimensional models should consider different aspects present in managing time-varying measures and dimensions.
- Typically DW data reflects measure changes leaving to application programming the representation of changes in dimension data. TDBs allow to express and manage changes for any data. The inclusion of the temporal support for DW data based on TDB research offers the solution for representing and managing in a similar way time-varying dimensions and measures.

Regarding temporal support in TDWs, most works include VT while some of them mention the possibility to have TT or BT support. However, they usually consider TT as the time when the fact is current in a DW, whereas in our model TT as well as VT are incorporated from source systems, if they exist and are required for analysis purposes. Further, in addition to TT and VT, we propose the inclusion of data warehouse loading time (DWLT) indicating since when the data has been current in TDWs.

On the other hand, even though some proposals formally describe the temporal support for a multidimensional model, to our knowledge none of them offer a graphical representation that can be used for communication between users and designers during the design phase of a TDW. Including temporal types in the conceptual model allows to include temporal semantics as an integral part of TDWs.

In this paper we introduce temporal extensions for the MultiDimER model (Malinowski & Zimányi 2005). Due to space limitations, we only refer to levels and hierarchies. Section 2 briefly recalls the main features of the MultiDimER model. Section 3 presents the temporal types included in the model. Section 4 describes our proposal for representing time-varying levels while Section 5 refers to changes occurring in levels as well as in relationships between them. Section 6 introduces a metamodel for a temporal dimension. In Section 7 we present a brief description of the transformation of the temporally extended MultiDimER model into the ER model. Finally, the related works are presented in Section 8 and conclusions are given in Section 9.

2 Overview of the MultiDimER model

It has been acknowledged for several decades that conceptual models are the best vehicle for communicating with users during the design process. We proposed the MultiDimER model (Malinowski & Zimányi 2004), a conceptual model based on ER constructs that allows to include different kinds of hierarchies. Figure 2 represents graphical notations used in our model.

²We ignore modifications due to errors during data loading and deletion for purging DW data.

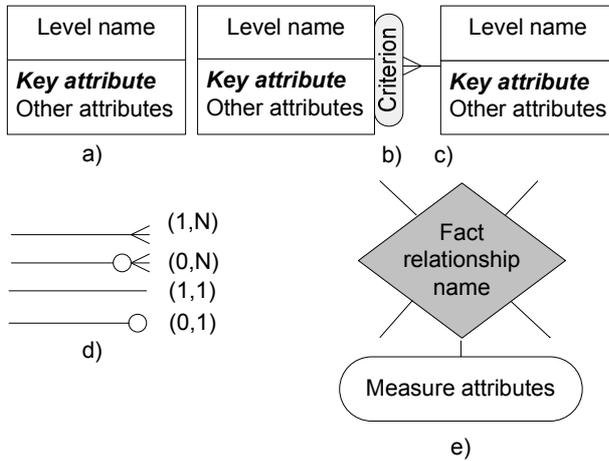


Figure 2: Notations of our multidimensional model: a) one-level dimension, b) analysis criterion, c) hierarchy, d) cardinalities, and e) fact relationship.

We briefly recall the definition of the MultiDimER model³. We define a *schema* as a finite set of dimensions and fact relationships. A *dimension* is an abstract concept for grouping data that shares a common semantic meaning within the domain being modelled. It represents either a level, or one or more hierarchies. Levels correspond to entity types (Figure 2 a). Every instance of a level is called a *member*.

Hierarchies are required for establishing meaningful paths for roll-up and drill-down operations. They express different structures according to the criteria used for analysis (Figure 2 b), e.g., geographical location or organizational structure. A hierarchy contains several related levels (Figure 2 c). *Cardinalities* (Figure 2 d) indicate the minimum and the maximum numbers of members in one level that can be related to a member in another level. Given two consecutive levels of a hierarchy, the higher level is called *parent* and the lower level is called *child*. A level of a hierarchy that does not have a child level is called *leaf*; the last level, i.e., the one does not have a parent level is called *root*. The root represents the most general view of data.

Levels contain one or several *key attributes* (represented in bold and italic in Figure 2) and may also have other *descriptive attributes*. A key attribute of a parent level defines how child members are grouped. A key attribute in a leaf level or in a level forming a dimension without hierarchy indicates the granularity of measures in the associated fact relationship.

A *fact relationship* (Figure 2 e) represents an n -ary relationship between leaf levels. It may contain attributes commonly called *measures*. The latter usually represent numerical data meaningful for leaf members that are aggregated while traversing a hierarchy. Since the roles of a fact relationship always have (0,N) cardinality, we omit such cardinalities to simplify the model.

3 Temporal types for TDW

Most works in TDWs, e.g., (Abelló & Martín 2003, Koncilia 2003) include valid time for representing when the data is valid in the modeled reality. This temporal type is also important for TDW applications since it allows to aggregate measures cor-

rectly as demonstrated in several works, e.g., (Eder et al. 2002).

Further, the usual practice for TDWs is to ignore TT coming from source systems, e.g., (Bliujute, Slivinskas & Jensen 1998, Body, Miquel, Bédard & Tchounikine 2003, Mendelzon & Vaisman 2000). However, in this way traceability applications, e.g., for fraud detection cannot be implemented. Other approaches, e.g., (Abelló & Martín 2003) transform TT from source systems to represent VT in the TDW. This is semantically incorrect because data may be included in databases after their period of validity has expired, e.g., client's previous address.

Moreover, some works, e.g., (Koncilia 2003), consider TT generated in a TDW in the same way as TT is used in TDBs, i.e., it allows to know when data was inserted, modified, or deleted from databases. Nevertheless, TDW data is neither modified nor deleted. Thus, TT in TDWs represents indeed the time when data was loaded into a TDW. This time is called in our model *data warehouse loading time* (DWLT).

DWLT can differ from TT or VT of source systems due to the delay between the time when the changes occurred in source systems and the time when these changes are integrated into a TDW. DWLT is important especially in active DWs (Bruckner & Tjoa 2002) and in creating TDWs from non-temporal sources (Yang & Widom 1998). It also may help to better understand decisions made in the past and adjust loading frequencies if necessary. For example, based on a growing tendency of product sales during weeks 10, 11 and 12 (Figure 3), it was decided to buy more products. However, only in the next DW load, occurred eight weeks later, a new situation has been revealed: a sudden decrease of sales. Thus, an additional analysis can be performed to understand the causes of these changes in sales behaviour. Further, the decision of more frequent loads may be taken.

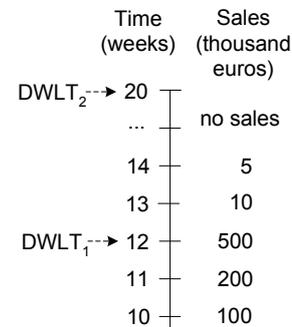


Figure 3: An example of the usefulness of having DWLT.

In some applications not only the validity of member attributes but also the member existence in the modelled reality is important, i.e., its lifespan (LS). The inclusion of LS allows to perform different analysis, e.g., discovering how sales change after the exclusion of some products.

Since current DWs do not offer different temporal types, users may have difficulties in expressing their needs for some kinds of applications, e.g., for fraud detection when TT from a source system is required. Our MultiDimER model meets users' expectations in modelling multidimensional data that vary over time allowing VT, TT, or bitemporal time (BT) coming from source systems and DWLT generated by a TDW. Further, if the source systems include LSs for data representing level members, this temporal type may also be included in a TDW.

In the modelling process, application requirements determine the type of temporal support (none, VT,

³The formal semantics of the MultiDimER model based on denotational specification is described in (Malinowski & Zimányi 2005).

TT, BT, DWLT) that needs to be captured in each element of a TDW (attributes, levels, hierarchies, and/or measures). Obviously that depends on whether or not the different data sources of the TDW provide temporal support. These sources may also contain user-defined time attributes playing the role of VT. Therefore, based on the classification of source systems given by Jarke *et al.* (2003) and additionally considering TDBs as another kind of a source system similar to Abelló and Martín (2003), in the following we discuss the temporal support that can be obtained from source systems while building TDWs.

- *Snapshot*: the access to the data in source systems is done through dump of data. To find the changes, the current data is compared with the previous snapshot(s). The time when the snapshot is realized does not determine neither TT nor VT. However, VT may be included as a user-defined attribute.
- *Queryable*: source systems offer a query interface. The detection of changes is done by periodic polling of the data in source systems and by comparing them with the previous version. They may be considered as snapshot systems with the difference of having direct access to data, thus they may contain VT as a user-defined attribute.
- *Logged*: all actions are registered. The periodic polling of data is required for discovering what kinds of changes to which data are applied. The log files contain TT that may be retrieved. Similar to the previous systems, VT may be present.
- *Specific*: each data is a particular case. There is not a general method for data extraction and detection of data changes. These systems can be considered as logged systems if either delta files or timestamps for attributes are available; otherwise, they may be treated as snapshot sources. Therefore, they may include TT and/or VT.
- *Callback or internal actions*: source systems provide triggers, active capabilities, or programming environment so they are able to automatically detect changes of interest and notify those changes to the interested parties, i.e., to a TDW. They offer TT and may include VT.
- *Replicated*: the detection of changes is done by analysing the messages sent by the replication system. This may happen manually, periodically, or using specific criteria. Depending on the features of the change monitor, this kind of systems may offer TT and VT.
- *Bitemporal*: TDBs include the information that allow to know when the objects are valid in reality and when they are current in a DB. Since bitemporal aspect is already represented in source systems, TT and VT are available.

In the following for simplifying the discussion we will refer to VT; the inclusion of TT is straightforward even though it is less used for dimensional data.

4 Time-varying levels

Changes in a level can occur either for attribute values (e.g., a product changes its ingredients) or for a member as a whole (e.g., inserting or deleting a product). For the former, we use attribute timestamping since it better represents reality keeping changes only for the specified attributes. For the latter, to indicate

the time when a member exists in the modelled reality, i.e., its lifespan, we use the LS symbol next to the level name. A level that includes temporal attributes and/or a lifespan support is called a *temporal level*.

Not all levels or their attributes need to represent changes in time. The MultiDimER model allows to choose which historical data users want to keep by including the symbols of corresponding temporal types for attributes and/or for a level. Notice that the changes to a level member as whole can be represented in the model independently of the fact that the level has temporal attributes.

Figure 4 a) shows an example of an Employee level that includes temporal attributes Position and Title. We group time-varying attributes firstly, to ensure that both kinds of attributes (temporal and non-temporal) can be clearly represented and secondly, to include a smaller number of symbols. For indicating the specific temporal types we use the abbreviations VT, TT, BT, and DWLT.

	Employee
	Employee id First name Last name Birth day Address
VT	Position Title

a)

LS	Employee
	Employee id First name Last name Birth day Address
VT	Position Title

b)

Figure 4: Representation of a) time-varying attributes and b) level lifespan.

The lifespan support for level members (Figure 4 b) indicates that each member includes the LS together with one value per attribute for non-temporal attributes and history of changes for temporal attributes. For example in Figure 4 b) every employee includes the LS together with one value per non-temporal attribute (e.g., Address) and the history of values for Position and Title.

Existing temporal models impose constraints for timestamped attributes and their corresponding object (entity) types, e.g., the VT of attribute values must be within the LS of the object (entity). Our model does not force it. In this way, different situations can be modelled, e.g., a product that does not belong to a store inventory (it is not included in the master file), but it is on sales for defining its acceptance level. For this product, the VT of temporal attributes may not be within the product LS. On the other hand, temporal integrity constraints may be explicitly defined, if required, using a calculus that includes Allen's operators (Allen 1984).

Further, the LS as well as VT used for attributes can be combined with TT or DWLT. In this way, users can obtain the information when the level member or the specific attributes values is current in a source or in a TDW, respectively.

5 Time-varying hierarchies

The MultiDimER model allows to represent hierarchies that contain several related levels. Given two consecutive levels in a hierarchy, the levels, the relationship between them or both the levels and the relationship between them may be temporal. We examine next these different situations.

5.1 Temporal levels and non-temporal relationships between them

Temporal levels can be associated with non-temporal relationships. Temporality in levels requires to keep changes for attributes or for lifespan of members. On the other hand, non-temporal relationships indicate that either these relationships never change or if they do, only the last modification is kept. An example is given in Figure 5.

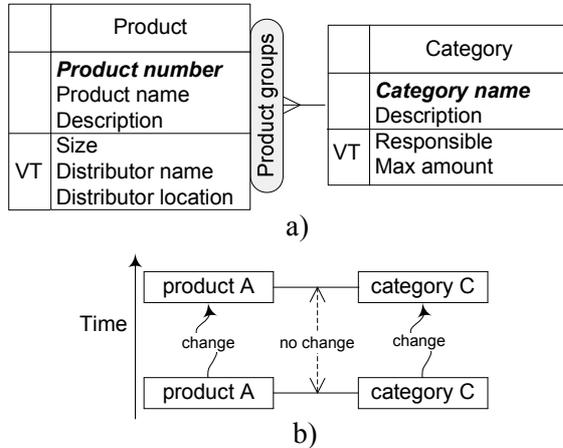


Figure 5: Time-varying levels forming a hierarchy: a) model and b) example of changes.

Nevertheless, if level members change the key attributes used for traversing from one level to another during the roll-up and drill-down operations, incorrect analysis scenario or dangling references may occur. For example, suppose in Figure 5 a) that the Product and Category levels include VTs for key attributes. Product A belongs to category C, but this category is divided into two new categories called C1 and C2, leaving category C invalid from now on. If the relationship product A – category C1 replaces a previous version of product A – category C, the analysis previous to this change will be incorrect, i.e., the measure will be aggregated to a new category C1 that did not exist prior to that change. If the relationship is not modified, references to the invalid version of category C will be made for the sales occurring after the categories have split.

To avoid an incorrect management of hierarchies as described above, we allow temporal levels with non-temporal relationships between them only for those levels that do not keep their LS and/or do not include VT for their key attributes. For example in Figure 5 a) the only changes allowed are those that do not affect relationships between members of these levels, e.g., a product changes its distributor but it belongs to the same category. Figure 5 b) illustrates which changes are allowed.

Notice that DWLT can always be included for levels or attributes since this temporal type only refers to the time when TDW members or attribute values are available for analysis purposes and this time does not affect their validity.

5.2 Temporal levels and temporal relationships between them

Temporal levels may include lifespan support and/or time-varying key attributes. This temporal support ensures to keep all changes occurring to level members and/or attribute values. However, these changes may affect the relationships with members of child and/or parent levels. For example, the geographical distribution in Europe during the last 20 years has changed

since some countries cease to exist, are merged, or split (Eder et al. 2002). As a consequence, in hierarchies representing this geographical distribution, reassignment of states or provinces to new countries may be required.

Therefore, in the case when levels include LS support and/or VT for key attributes, to avoid incorrect management of hierarchies as described in the previous section, relationships between temporal levels must also be temporal. This restriction in the MultiDimER model as in most TDB models helps to avoid dangling references, i.e., linking to non-existing objects.

The MultiDimER model allows to include VT, TT, BT, and/or DWLT for representing time-varying relationships between levels. We do not include LS for these relationships since these relationships do not exist without their participating levels.

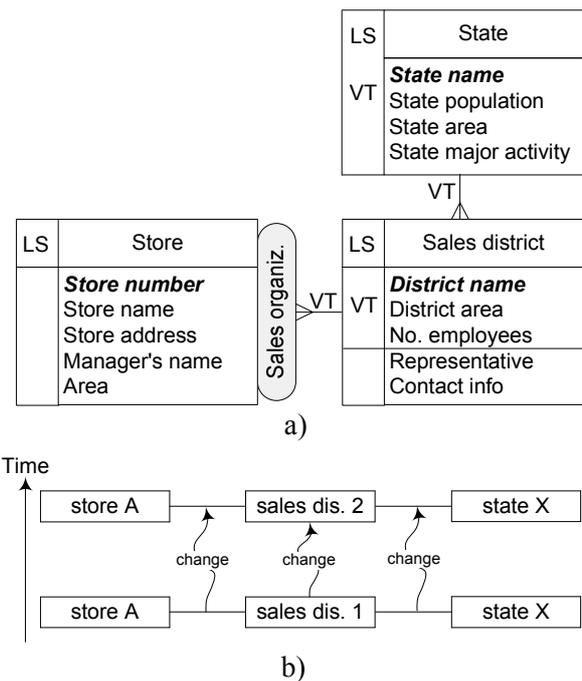


Figure 6: Time-varying levels and time-varying relationships between them: a) model and b) example of changes.

In the example of Figure 6 a) all levels are temporal. They include LS for dimension levels and VT for some attributes. Further, the relationships between levels are also temporal. Suppose that the sales company is in an active development and changes to Sales districts may occur to improve the organizational structure. These changes may affect the relationship with members of the Store and State levels (Figure 6 b).

Further, the constraint for relationships between levels forming a hierarchy is more restrictive than the one usually used for relationships between temporal objects. In TDBs the valid time of a relationship instance must be included in the intersection of the valid times of participating objects. In multidimensional hierarchies it is further required that every valid child (respectively parent) member must be associated with at least one valid parent (respectively child) member in order to ensure correctness of the roll-up and drill-down operations. Validity can refer to the LS of a level as well as to the VT of key attributes leading to the following constraints:

1. Every time point included in the LS of a level must be included in the LS of some member of

the next level, i.e., a valid child member must have a valid parent member and vice versa. If this condition is not fulfilled, structural changes to hierarchies could occur, e.g., forcing some level members to skip the current parent level⁴.

2. Every time point included in the VT of a key attribute (i.e., used for aggregation purposes) of a child (respectively parent) member must be included in the VT of some key attribute of a parent (respectively child) member.

5.3 Non-temporal levels and temporal relationships between them

Non-temporal levels can be linked with temporal relationships if the values of the level members do not change but the changes to relationship between levels are kept. Some examples of this changing relationships, called *transitions* (Zimányi, Parent & Spaccapietra 1997) include *evolution* and *extension*. The former occurs when a child member ceases to be related to one parent member and is assigned to another one, e.g., a section is assigned to a new division (Figure 7 a). The latter takes place when a child member belongs to the original parent member and additionally a new relationship with a different parent member is included, e.g., the old section is assigned to the new division, leaving it also as a part of the old division (Figure 7 b).

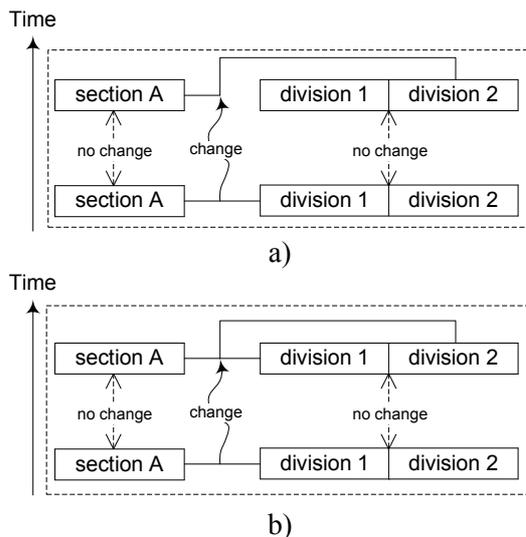


Figure 7: Time-varying relationships between non-temporal levels: a) evolution and b) extension.

To represent temporal support allowing changes in relationships between non-temporal levels we place the corresponding symbol, e.g., VT, on the link between hierarchy levels. To ensure consistency during roll-up and drill-down operations, level members cannot be modified as explained in the previous sections.

5.4 Conditions for including a temporal support in hierarchies

Based on the explanations given in the previous sections, in this section we will summarize the conditions for including temporal support in multidimensional hierarchies. Further, we also include cases not seen until now when either temporal or non-temporal

⁴We do not consider structural changes to hierarchies since they require schema versioning that is out of the scope of this paper.

relationships exist between a temporal and a non-temporal levels.

1. Temporal levels and non-temporal relationships between them: temporal features can only be applied for attributes that do not participate in the roll-up and drill-down operations.
2. Temporal levels and temporal relationships between them: levels, attributes, and links indicating the relationship between levels can be temporal.
3. Non-temporal levels and temporal relationships between them: no modifications to level members are allowed.
4. Non-temporal levels and non-temporal relationships between them: changes to dimension data cannot be kept. This is the current situation where implementation “tricks” must be used to represent changes to level members and/or to relationships between them.
5. One temporal and one non-temporal level and temporal relationships between them: similar to Case 3, thus non-temporal level members cannot be modified.
6. One temporal and one non-temporal level and non-temporal relationships between them: similar to Case 1, thus temporal level members only can have temporal types for non-key attributes.

5.5 Snapshot and lifespan cardinalities

Cardinalities in a non-temporal model indicate the number of members of one level that can be related to member(s) of another level. In our model, this cardinality may be interpreted in two possible ways: the *snapshot cardinality* and the *lifespan cardinality*. The former is considered for every time instant while the latter over its lifespan.

The lifespan cardinality may be different from the snapshot cardinality because of the changes in hierarchies, both in levels and in relationships between them. Thus, when these temporal changes must be kept, both lifespan and snapshot cardinalities may be considered.

In the MultiDimER model the snapshot cardinality is by default equal to the lifespan cardinality; however, if these cardinalities are different, a dotted line with the LC symbol is inserted and it indicates the lifespan cardinality as shown in Figure 8.

Further, the constraint imposed on the cardinalities requires the minimum value as well as the maximum value of the lifespan cardinalities to be equal or greater than minimum and maximum values of the snapshot cardinalities, respectively.

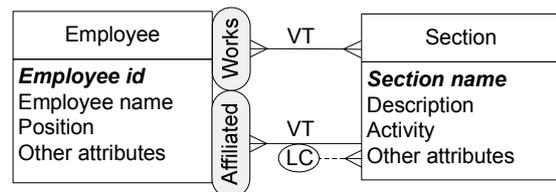


Figure 8: Snapshot and lifespan cardinalities between hierarchy levels.

In the example in Figure 8, the employee snapshot and lifespan cardinalities for the hierarchy Works are many-to-many indicating that an employee can work in more than one section at the same time instant

and over his lifespan. On the other hand, the snapshot cardinality for the hierarchy Affiliated is one-to-many, and the lifespan cardinality is many-to-many indicating that in every time instant an employee can be affiliated only to one section, but over his lifespan he can be affiliated to many sections.

6 Metamodel of a temporally-extended dimension in the MultiDimER model

We give next a metamodel for a dimension of the temporally-extended MultiDimER model - a conceptual model used to represent dimensions, hierarchies, and levels with attributes, which may change over time.

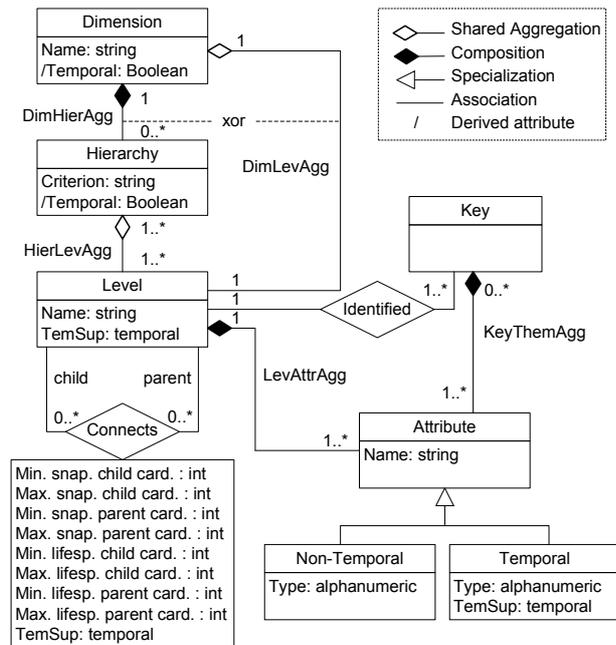


Figure 9: Metamodel of a dimension.

As shown in Figure 9, a dimension is comprised of either a level, or one or more hierarchies. A hierarchy contains several related levels. These levels are associated through child-parent relationship. Levels include attributes, some of which are key attributes used for aggregation purposes.

We define a *temporal level* as a level for which the application needs to keep its time-varying characteristics. This is captured by including different temporal types for attributes and/or for a level, i.e., the TempSup attribute in Figure 9. We allow VT, TT, or BT coming from source systems (if available) and DWLT generated by DBMS of a TDW. Notice, that VT for a level is represented by its LS.

Further, the relationship between levels may also be temporal independently of whether the levels are temporal or not. This is indicated in Figure 9 by the attribute called TempSup for the Connects relationship between child and parent levels. The model allows to include VT, TT, BT, and/or DWLT for this relationship.

Additionally, the relationship between two levels is characterized by cardinalities, which indicate the minimum and the maximum number of members in one level that can be related to a member in another level. We distinguish the snapshot cardinality and lifespan cardinality. The former is the cardinality in a instant of time whereas the latter represents this cardinality over members lifespan.

A dimension is temporal if it has at least one temporal hierarchy. A hierarchy is temporal if it has at

least one temporal level or one temporal relationship between levels. Since temporal hierarchies (respectively dimensions) can combine temporal and non-temporal levels (respectively hierarchies), we call a hierarchy (respectively dimension) *fully temporal* when all its levels and relationships between them (respectively hierarchies) are temporal. It is called *partly temporal* when it contains at least one non-temporal level or one non-temporal relationship between levels (respectively one non-temporal hierarchy).

7 Transformations to the ER model

The MultiDimER model can be implemented by mapping its specifications into those of operational data models. We already proposed the mapping into relational (Malinowski & Zimányi 2005) and object-relational (Malinowski & Zimányi 2006) data models. Further, another two-phase approach may be adopted where a MultiDimER schema is transformed into a conventional ER schema and afterwards, into a logical schema.

In this section, we briefly describe and give examples of mapping the temporally-extended MultiDimER model into the ER model. The ER model is a widely-used and platform-independent conceptual model. In addition, well-known rules for transformation of ER model into relational model exist, e.g., (Elmasri & Navathe 2003).

The MultiDimER model is mainly based on the ER constructs with their usual semantics, i.e., entity types, attributes, relationship types. Some additional semantics is provided for different kinds of hierarchies (Malinowski & Zimányi 2004) that is beyond the scope of this paper.

In the MultiDimER model a level corresponds to an entity type in the ER model. Non-temporal attributes are represented as monovalued attributes. The temporal support in the MultiDimER model is added in an implicit manner (Gregersen & Jensen 1998), i.e., the timestamp attributes used for capturing a temporal aspect are hidden using instead pictograms. Therefore, the transformation of the time-related data into classical non-temporal structures of the ER model requires additional attributes for timestamps.

The different temporal types can be represented in the ER model as follows: (1) a simple composite attribute for a period (Figure 10 a), (2) a multivalued composite attribute for a set of periods (Figure 10 b), (2) a monovalued attribute for an instant (Figure 10 c), and (3) a multivalued attribute for a set of instants (Figure 10 d). Notice that a set of periods or instants are used when the attribute has the same value in different periods or instants of time.

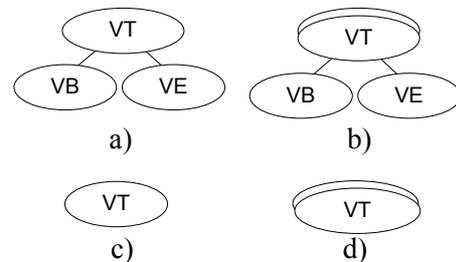


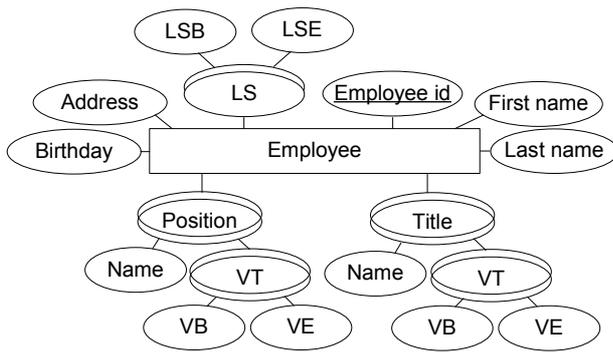
Figure 10: Different representations of VT in the ER model.

Figure 11 shows an Employee level and its corresponding ER diagram. The mapping of each temporal attribute requires a multivalued composite attribute;

it includes an attribute for which the temporal type is attached (Name for Position and Title attributes in Figure 11 b) and an additional attribute for a temporal type. The latter is represented in Figure 11 b) using a set of periods indicating that an employee can have the same position or title in different periods of time.

LS	Employee
	Employee id
	First name
	Last name
	Birth day
	Address
VT	Position
	Title

a)



b)

Figure 11: A level with temporal attributes: a) the MultiDimER model and b) corresponding ER model.

Further, as explained before, the MultiDimER model can represent temporal changes to a level member as whole. This is expressed using the LS symbol next to the level name (Figure 11 a). This temporal support is mapped into the ER model using an additional multivalued composite attribute containing the begin (LSB) and the ending (LSE) instants of the lifespan (Figure 11 b). We used a temporal element, i.e., a set of periods since this allows to represent discontinuous lifespan, e.g., a professor leaving for sabbatical during some period of time.

Relationships linking the levels of a hierarchy in the MultiDimER model are usual binary relationships in the ER model. Since this relationship in our model can be temporal, the corresponding binary relationship in the ER model should include an attribute (or several depending on the applied temporal support) in a similar way as was explained for time-varying attributes of a level. For example, the mapping of a temporal relationship between Store and Sales district levels for the example in Figure 6 is shown in Figure 12⁵.

In this case, the snapshot and lifespan cardinalities are the same, i.e., many-to-one. If these cardinalities are different, the highest cardinalities is mapped to the ER model; according to the previously specified constraint in Section 5.5, the lifespan cardinality will be mapped.

As can be seen in Figure 11 and 12, the MultiDimER model provides better conceptual representation of time-varying attributes, levels, and relationships. It contains less elements, it clearly allows to distinguish which data changes should be kept, and it

⁵For simplicity we do not present level attributes in the figure.

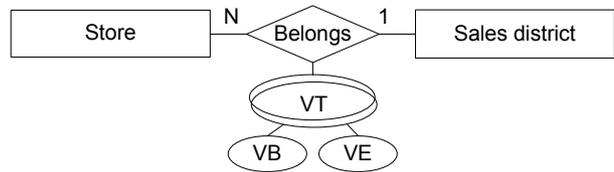


Figure 12: ER representation of temporal relationships between levels in the MultiDimER model.

leaves outside of user's concerns some more technical aspects such as multivalued or composite attributes.

8 Related work

The necessity to manage time-varying data has been acknowledged for several decades, e.g., (Snodgrass 1995). However, no such consensus has been reached for representing time-varying multidimensional data considering the particularities of DW semantics.

Works related to TDWs raise many issues, e.g., the inclusion of temporal types in TDWs, e.g., (Abelló & Martín 2003, Bruckner & Tjoa 2002), temporal querying of multidimensional data, e.g., (Mendelzon & Vaisman 2003, Pedersen, Jensen & Dyreson. 2001), correct aggregation in presence of data and structural changes, e.g., (Eder et al. 2002, Hurtado, Mendelzon & Vaisman 1999, Mendelzon & Vaisman 2003), temporal view materialization from non-temporal sources, e.g., (Yang & Widom 1998), evolution of a multidimensional structure, e.g., (Body et al. 2003, Eder et al. 2002, Mendelzon & Vaisman 2003), or implementation considerations for a temporal star schema, e.g., (Bliujute et al. 1998).

In the following we refer to works that (1) propose different temporal types for TDWs and (2) offer conceptual models for TDWs.

The inclusion of different temporal types in TDWs is briefly mentioned in several works. Most of them consider VT (Body et al. 2003, Bliujute et al. 1998, Eder et al. 2002, Mendelzon & Vaisman 2003, Ravat & Teste 2000, Yang & Widom 1998); some of them mention that it is easy to incorporate TT without giving a deeper analysis (Mendelzon & Vaisman 2003, Pedersen et al. 2001). Other authors consider TT generating it either in a TDW (Abelló & Martín 2003, Koncilia 2003) or of in a given source system (Bruckner & Tjoa 2002)⁶.

A more extensive analysis of temporal types for TDWs is given by Abelló and Marín (2003). They discuss the inclusion of TT and VT in TDWs taking into account different types of sources that integrate data in a TDW. VT is calculated based on TT of either a DW or a source. The exception is made for sources based on TDBs, considering only one temporal type, e.g., VT and converting another one, e.g., TT into a user-defined attribute. However, Abelló and Marín (2003) do not consider possible existence of user-defined time attributes in source systems, which may serve for establishing VT in TDWs. Also, TT from source systems is ignored or transformed for representing VT in TDWs. We do not consider TT as a possible approximation of VT, since data can be included and be current in DBs after its validity has expired, e.g., courses taught five years ago.

Therefore, even though most works include VT and some mention the possibility to have TT or explicitly present BT support, they usually considered TT as time where a fact is current in DW, whereas in our model TT as well as VT are incorporated from

⁶It is called *revelation time* in (Bruckner & Tjoa 2002).

source systems. Further, only Bruckner and Tjoa (2002) discuss the inclusion of VT, TT, and DWLT for active data warehouses, however, they do not offer a conceptual model for a TDW that includes these temporal types.

Several works are dedicated to conceptual modelling of TDWs. Body *et al.* (2003) define a conceptual TDW model that allows a member to have several valid member versions for a given time (when VT overlaps). Further, they include a temporal relationship that establishes an explicit link between two member versions and represents the roll-up function. Since a dimension is a set of member versions and a set of temporal relationships between these members, a temporal dimension is considered as a directed graph where nodes are member versions and arcs are relationships.

Eder *et al.* (2002) propose a temporal multidimensional model called COMET; it allows to represent changes at the schema and instance levels. The model includes VT for members and for relationships between members forming hierarchies. They include a list of constraints to ensure the integrity of their model. Further, in order to reduce incorrect OLAP results due to the dimension changes, their model includes transformation functions given by the user. The COMET model was extended by Koncilia (2003) including TT of a TDW.

Mendelzon and Vaisman (2003) propose a temporal multidimensional model that reuses results from the TDB community. They formally define temporal dimension schema and instances as well as a temporal fact table, which are used for defining temporal multidimensional database. Using VT they build a TO-LAP query language that allows the user to choose the way data should be aggregated.

Further, Pedersen *et al.* (2001) extend the basic multidimensional model by temporal support. Their model allows the inclusion of VT as well as TT. These temporal types can be used to express changes in dimension members including their representation, in hierarchy links, and in fact-dimension relationships.

On the other hand, Ravat and Teste (2000) define a DW model using object-oriented approach; it allows to integrate temporal and archive data. Temporal data are used for storing the detailed data evolution while archive data store the summarized data evolutions.

In general, these models formally describe the temporal support for multidimensional models, allowing to express changes in dimension members, hierarchy links, and in fact relationships. However, none of them offer a graphical representation based on a multidimensional view of temporal data that can be used for communication between users and designers. Further, they do not consider different aspects as proposed in this work, e.g., a hierarchy that may have temporal and non-temporal levels linked with either temporal or non-temporal relationships.

9 Conclusions

Bringing together two research areas, Data Warehouses (DWs) and Temporal Databases (TDBs), allows to combine the achievements of each of them leading to the emerging field of Temporal Data Warehouses (TDWs). Nevertheless, neither DWs nor TDBs have a well-accepted conceptual model that can be used for capturing users' requirements.

In this paper, we proposed a temporal extension of the MultiDimER model for representing time-varying levels and hierarchies. This model allows to represent both temporal and time-invariant levels and hierarchies. In this way, users and designers are able to

choose and express in a unambiguous way which elements they want to be time invariant and for which data they want to express changes occurred in time, as recommended by Gregersen and Jensen (1998).

We included in the model valid and transaction time coming from source systems and the data warehouse loading time generated by a TDW. In this way, users can traverse hierarchies knowing when data is valid in the modelled reality, expand the analysis to traceability applications, and know since when data has been available in a TDW. Further, having lifespan for level members allows to know how the exclusion or inclusion of different members may affect measure values.

However, the inclusion of different temporal types depends on users' requirements and also on their availability in source systems. Taking into account different kinds of source systems, we discussed which temporal support they can offer.

Next, we proposed to extend the MultiDimER model adding time-varying attributes and lifespan of a level. We also discussed three different cases for time-varying hierarchies: (1) temporal levels with non-temporal relationships between them, (2) temporal relationships between temporal levels, and (3) temporal relationships between non-temporal levels. In the first case, we did not allow the modification for key attributes participating in the roll-up and drill-down operations. For the second case we established constraints ensuring that every valid child (respectively parent) member is related to its valid parent (respectively child) member. In this way, changes to members as well as the relationship between them are considered avoiding dangling references. For the third case we imposed the restriction that level members cannot be modified. Further, we included in the model the snapshot and lifespan cardinalities indicating the number of members of one level that can be related to members of another level in every time instant and over its lifespan, respectively.

Finally, we presented the metamodel of dimensions where levels as well as relationships between them may vary over time. We finished describing the transformation of the constructs of the MultiDimER model into the ER model.

Proposing the inclusion of temporal types in a conceptual model allows to include temporal semantics as an integral part of TDWs. Afterwards, logical and physical models can be derived from such a conceptual representation. Further, these logical models can be obtained using a direct mapping of the MultiDimER constructs (Malinowski & Zimányi 2005, Malinowski & Zimányi 2006) or using a two-phase approach: first to the well-known ER model and then to a chosen logical representation.

This work belongs to a larger project aiming at developing a methodology for conceptual design of spatio-temporal data warehouses.

References

- Abelló, A. & Martín, C. (2003), A bitemporal storage structure for a corporate data warehouse, *in* 'Proc. of the 5th Int. Conf. on Enterprise Information Systems', pp. 177–183.
- Allen, J. (1984), 'Towards a general theory of action and time', *Artificial Intelligence* **23**(2), 123–154.
- Bliujute, R., Slatenis, S., Slivinskas, G. & Jensen, C. (1998), Systematic change management in dimensional data warehousing, Technical report, Time Center, TR-23.

- Body, M., Miquel, M., Bédard, Y. & Tchounikine, A. (2003), Handling evolution in multidimensional structures, in 'Proc. of the 19th Int. Conf. on Data Engineering', pp. 581–592.
- Bruckner, R. & Tjoa, A. (2002), 'Capturing delays and valid times in data warehouses – towards timely consistent analyses', *Journal of Intelligent Information Systems* **19**(2), 169–190.
- Eder, J., Koncilia, C. & Morzy, T. (2002), The COMET metamodel for temporal data warehouses, in 'Proc. of the 14th Int. Conf. on Advanced Information Systems Engineering', pp. 83–99.
- Elmasri, R. & Navathe, S. (2003), *Fundamentals of Database Systems*, fourth edn, Addison-Wesley.
- Elmasri, R. & Wu, G. (1990), A temporal model and query language for ER databases, in 'Proc. of the 6th Int. Conf. on Data Engineering', pp. 76–83.
- Gregersen, H. & Jensen, C. (1998), Conceptual modeling of time-varying information, Technical report, Time Center, TR-35.
- Hurtado, C., Mendelzon, A. & Vaisman, A. (1999), Maintaining data cubes under dimension updates, in 'Proc. of the 15th Int. Conf. on Data Engineering', pp. 346–355.
- Inmon, W. (2002), *Building the Data Warehouse*, John Wiley & Sons.
- Jarke, M., Lenzerini, M., Vassiliou & Vassiliadis, P., eds (2003), *Fundamentals of Data Warehouse*, Springer.
- Kimball, R., Ross, M. & Merz, R. (2002), *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*, John Wiley & Sons.
- Koncilia, C. (2003), A bi-temporal data warehouse model, in 'Proc. of Short Papers of the 15th Int. Conf. on Advanced Information Systems Engineering', pp. 77–80.
- Malinowski, E. & Zimányi, E. (2004), OLAP hierarchies: A conceptual perspective, in 'Proc. of the 16th Int. Conf. on Advanced Information Systems Engineering', pp. 477–491.
- Malinowski, E. & Zimányi, E. (2005), Hierarchies in a multidimensional model: from conceptual modeling to logical representation. Accepted for publication in *Data & Knowledge Engineering*.
- Malinowski, E. & Zimányi, E. (2006), Object-relational representation of a conceptual model for temporal data warehouses. Submitted to publication.
- Martín, C. & Abelló, A. (2003), A temporal study of data sources to load a corporate data warehouse, in 'Proc. of the 5th Int. Conf. on Data Warehousing and Knowledge Discovery', pp. 109–118.
- Mendelzon, A. & Vaisman, A. (2000), Temporal queries in OLAP, in 'Proc. of the 26th Very Large Database Conference', pp. 243–253.
- Mendelzon, A. & Vaisman, A. (2003), Time in multidimensional databases, in M. Rafanelli, ed., 'Multidimensional Databases: Problems and Solutions', Idea Group Publishing, pp. 166–199.
- Pedersen, T., Jensen, C. & Dyreson, C. (2001), 'A foundation for capturing and querying complex multidimensional data', *Information Systems* **26**(5), 383–423.
- Ravat, F. & Teste, O. (2000), A temporal object-oriented data warehouse model, in 'Proc. of the 11th Int. Conf. on Database and Expert Systems', pp. 583–592.
- Snodgrass, R., ed. (1995), *The TSQL2 Temporal Query Language*, Kluwer Academic Publishers.
- Yang, J. & Widom, J. (1998), Maintaining temporal views over non-temporal information source for data warehousing, in 'Proc. of the 6th Int. Conf. on Extending Database Technology', pp. 389–403.
- Zimányi, E., Parent, C. & Spaccapietra, S. (1997), TERC+: a temporal conceptual model, in 'Proc. of the Int. Symp. on Digital Media Information'.

Visualization of Music Impression in Facial Expression to Represent Emotion

Takafumi Nakanishi

Takashi Kitagawa

Graduate School of Systems and Information Engineering
University of Tsukuba,
Tsukuba, Ibaraki 305-8573, Japan
Email: takafumi@mma.cs.tsukuba.ac.jp
takashi@cs.tsukuba.ac.jp

Abstract

In this paper, we propose a visualization method of music impression in facial expression to represent emotion. We apply facial expression to represent the complicated and mixed emotions. This method can generate facial expression corresponding to impressions of music data by measurement of relationship between each basic emotion for facial expression and impressions extracted from music data. The feature of this method is a realization of an integration between music data and the facial expression that convey various emotions effectively. One of the important issues is a realization of communication media corresponding to human Kansei with less difficulty for a user. Facial expression can express complicated emotions with which various emotions are mixed. Assuming that an integration between existing mediadata and facial expression is possible, visualization corresponding to human Kansei with less difficulty realized for a user.

Keywords: Mediadata, Facial Expression, Music Data, Impression, Kansei.

1 Introduction

A large amount of information resources have been distributed in wide area networks. In this environment, a current interface, for example, computer keystrokes, is difficult of computer manipulation for human being. One of the important issues is a realization of communication media corresponding to human Kansei with less difficulty for a user. The concept of "Kansei" includes several meanings on sensitive recognition, such as "impression," "human senses," "feelings," "sensitivity," "psychological reaction" and "physiological reaction."

Generally, it is important to understand each other emotion correctly in our communication. In particular, facial expression is important as media which convey various emotions effectively. The facial expression is one of nonverbal behaviors. The facial expression can express complicated emotions with which various emotions are mixed which cannot be expressed with words.

The researches which realize composition and recognition of the facial expression are done actively. In these researches, Facial Action Coding System (FACS)(Ekman & Friesen 1978, Ekman & Friesen 1987) is used strictly and most widely. FACS

describes facial expression with the combination of some Action Units (AU's). AU's are the minimum units of facial expression operations which are visually discernible. These research results have shown the combination of AU's for expressing 6 basic emotions, which are "happiness", "surprise", "fear", "anger", "disgust", and "sadness". The combination of these basic emotions can express complicated facial expression.

There are the followings as previous researches on construction of facial expression, research which mounts AU and creates the picture of expression(Choi, Harashima, & Takebe 1990), research of facial imitation by 3D face robot agent(Hara, & Kobayashi 1996), etc. These researches realize a construction of facial expression which is close to an actual expression.

In this paper, we propose a visualization method of music impression in facial expression to represent emotion. We apply facial expression to represent the complicated and mixed emotions. This method can generate facial expression corresponding to impressions of music data by measurement of relationship between each basic emotion for facial expression and impressions extracted from music data.

We have already proposed a semantic associative search method based on a mathematical model of meaning(Kitagawa & Kiyoki 1993, Kiyoki, Kitagawa & Hayama 1994). This model is applied to extract semantically related words by giving context words. This model can measure the relation between each word, mediadata, and so on.

In addition, we have already proposed a media-lexicon transformation operator for music data(Kitagawa & Kiyoki 2001, Kitagawa, Nakanishi & Kiyoki 2004). This operator can extract meta-data which represents the impression of music data as weighted words.

This proposal method can generate facial expression corresponding to impression of music data utilizing the mathematical model of meaning and the media-lexicon transformation operator for music data. The feature of this method is a realization of an integration between existing mediadata, that is music data, and nonverbal behaviors that convey various emotions effectively, that is facial expression. Namely, the purpose of this method is different from the conventional methods.

The system using facial expression can express impressions of mediadata more appropriately compared with the system only using the information such as words. The facial expression can express complicated emotions. Assuming that complicated emotion can be expressed, human and the system can share mutual emotion and the interface corresponding to human Kansei can be realized.

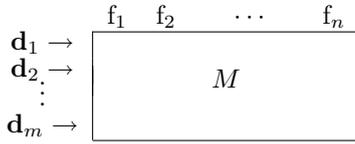


Figure 1: Representation of metadata items by matrix M

2 Mathematical Model of Meaning

The mathematical model of meaning (Kitagawa & Kiyoki 1993, Kiyoki, Kitagawa & Hayama 1994) provides semantic functions for computing specific meanings of words which are used for retrieving mediadata unambiguously and dynamically. The main feature of this model is that the semantic associative search is performed in the orthogonal semantic space. For details, see references (Kitagawa & Kiyoki 1993, Kiyoki, Kitagawa & Hayama 1994).

The mathematical model of meaning consists of:

1. Creation of a metadata space MDS

Create an orthonormal space for mapping the mediadata represented by vectors (hereafter, this space is referred to as the metadata space MDS). The specific procedure is shown below.

When m data items for space creation are given, each data item is characterized by n features (f_1, f_2, \dots, f_n). For given $\mathbf{d}_i (i = 1, \dots, m)$, the data matrix M (Figure 1) is defined as the $m \times n$ matrix whose i -th row is \mathbf{d}_i . Then, each column of the matrix is normalized by the 2-norm in order to create the matrix M .

- (a) The correlation matrix $M^T M$ of M is computed, where M^T represents the transpose of M .
- (b) The eigenvalue decomposition of $M^T M$ is computed.

$$M^T M = Q \begin{pmatrix} \lambda_1 & & & \\ & \ddots & & \\ & & \lambda_\nu & \\ & & & 0 \dots 0 \end{pmatrix} Q^T, \quad (1)$$

$$0 \leq \nu \leq n.$$

The orthogonal matrix Q is defined by

$$Q = (\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n) \quad (2)$$

where \mathbf{q}_i 's are the normalized eigenvectors of $M^T M$. We call the eigenvectors "semantic elements" hereafter. Here, all the eigenvalues are real and all the eigenvectors are mutually orthogonal because the matrix $M^T M$ is symmetric.

- (c) Defining the metadata space MDS

$$MDS := span(\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_\nu). \quad (3)$$

which is a linear space generated by linear combinations of $\{\mathbf{q}_1, \dots, \mathbf{q}_\nu\}$. We note that $\{\mathbf{q}_1, \dots, \mathbf{q}_\nu\}$ is an orthonormal basis of MDS .

2. Representation of mediadata in n-dimensional vectors

Each mediadata is represented in the n-dimensional vector whose elements correspond to

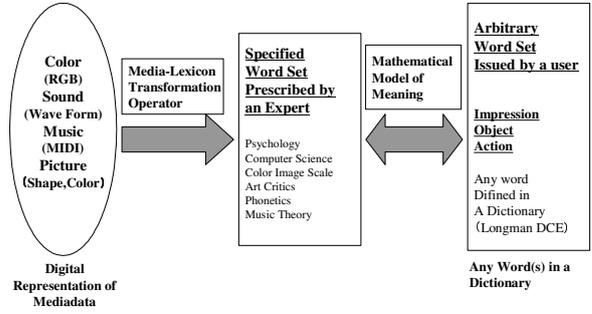


Figure 2: A framework of media-lexicon transformation operator.

n features. The specific procedure is shown below.

A metadata for mediadata P is represented in t weighted impression words $\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_t$. These impression words are extracted from media-lexicon transformation operator shown in section 3.

$$P = \{\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_t\}. \quad (4)$$

Each impression word is defined as an n dimensional vector by using the same features as the features of the data matrix M .

$$\mathbf{o}_i = (f_{i1}, f_{i2}, \dots, f_{in}) \quad (5)$$

The weighted impression words $\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_t$ are composed to form the mediadata vector, which is represented as an n dimensional vector. The Kansei operator shown in subsection 6 of section 4.2 realizes this composition. The mediadata is represented as mediadata vector which is n dimensional vector by using same features as the features of the data matrix M .

3. Mapping a mediadata vector into the metadata space MDS

A mediadata vector which is represented in n -dimensional vectors is mapped into the metadata space MDS by computing the Fourier expansion for a mediadata vector and semantic elements.

4. Semantic associative search

A set of all the projections from the metadata space MDS to the invariant subspaces (eigenspaces) is defined. Each subspace represents a phase of meaning and it corresponds to a context. A subspace of the metadata space MDS is selected according to the context. An association of a mediadata is measured in the selected subspace.

3 Media-lexicon Transformation Operator

In this section, we introduce a media-lexicon transformation operator \mathcal{ML} (Kitagawa & Kiyoki 2001).

3.1 A Framework of Media-lexicon Transformation Operator

In Figure 2, we show a framework of the media-lexicon transformation operator \mathcal{ML} (Kitagawa & Kiyoki 2001).

\mathcal{ML} is an operator which represents a relation between mediadata and some group of word sets given

by a research work by an expert of a specific disciplinary area. The operator ML is defined as

$$ML(Md) : Md \mapsto Ws$$

where, Md is an expression of mediadata and Ws is a specific set of words or a collection of word sets usually with weights. The mediadata Md is a specific expression of the mediadata usually in a digital format. The word set Ws is selected by an expert to express impression of the specific media.

By this operator ML , we can search or retrieve the mediadata by arbitrary words issued as a query, using the mathematical model of meaning (Kitagawa & Kiyoki 1993, Kiyoki, Kitagawa & Hayama 1994) which relates any given words to certain word groups dependent on the given context.

3.2 Media-lexicon Transformation Operator for Music Data

Media-lexicon transformation operator for music data (Kitagawa & Kiyoki 2001, Kitagawa, Nakanishi & Kiyoki 2004) extracts some impression words from music data. This operator extracts impression words of a song from elements (musical elements) that determine the form or structure of the song such as harmony, melody, and so on. The fundamental psychological research that examined correlation relationships between impressions and musical elements was conducted by Hevner (Hevner 1935, Hevner 1936, Hevner 1937, Umemoto.ed. 1966). This operator uses the correlation relationships indicated by Hevner to calculate correlations between these sets of musical elements and impression words.

3.2.1 Research of Hevner

In Hevner's research (Hevner 1935, Hevner 1936, Hevner 1937, Umemoto.ed. 1966), key, tempo, pitch, rhythm, harmony, and melody were given as musical elements. Hevner examined the correlation relationships between these 6 musical elements and 8 categories of impression words (Figure 3). Each category of impression words was created by collecting together impression words that had similarities to other words in that category. The 8 categories of impression words were further arranged in a circle so that categories were adjacent to other categories to which they had similarities. Hevner experimentally obtained correlation relationships between musical elements and impressions represented by categories of impression words.

3.2.2 An Implementation Method of Media-lexicon Transformation Operator for Music Data

This section shows an implementation method of media-lexicon transformation operator for music data. For details, see references (Kitagawa & Kiyoki 2001, Kitagawa, Nakanishi & Kiyoki 2004).

This operator consists of three steps:

Step 1 : Composition of Transformation Matrix T .

Transformation Matrix T shown in Figure 4 is composed by the correlation between musical elements and each impression which are given by Hevner.

Step 2 : Extraction of musical element vector \mathbf{s} .

A musical element analysis data consisting of data for the structure and form of the song is extracted from a *Standard MIDI File (SMF)* as digitized music data. We form a musical element vector \mathbf{s} which consists of music elements *key, tempo, pitch, rhythm, harmony* and

			c6 bright cheerful gay happy joyous merry	c5 delicate fanciful graceful humorous light playful quaint sprightly whimsical	
c7 agitated dramatic exciting exhilarated impetuous passionate restless sensational soaring triumphant					c4 calm leisurely lyrical quiet satisfying serene soothing tranquil
c8 emphatic exalting majestic martial ponderous robust vigorous					
	c1 awe-inspiring dignified lofty sacred serious sober solemn spiritual			c3 dreamy longing plaintive sentimental	
		c2 dark depressing doleful frustrated gloomy heavy melancholy mournful pathetic sad tragic			

Figure 3: Hevner's 8 categories of impression words.

	key'	tempo'	pitch'	rhythm'	harmony'	melody'
c ₁	4	-14	-10	18	3	4
c ₂	-12	-12	-19	3	-7	0
c ₃	-20	-16	6	-9	4	0
c ₄	3	-20	8	-2	10	3
c ₅	21	6	16	8	12	-3
c ₆	24	20	6	-10	16	0
c ₇	0	21	-9	2	-14	-7
c ₈	0	6	-13	10	-8	-8

Figure 4: Transformation Matrix T indicating the relationships between impression word categories and musical elements.

melody generated from the musical element analysis data.

The vector is represented as follows.

$$\mathbf{s} = (\text{key}, \text{tempo}, \text{pitch}, \text{rhythm}, \text{harmony}, \text{melody})^t. \quad (6)$$

Step 3 : Extraction of impression words

Transformation Matrix T transforms musical element vector \mathbf{s} to the weights \mathbf{v} (music category vector) of the 8 categories of impression words.

$$\mathbf{v} = T\mathbf{s} \quad (7)$$

A music category vector \mathbf{v} is an 8 dimensional real valued vector.

$$\mathbf{v} = (v_{c_1}, v_{c_2}, \dots, v_{c_8})^t. \quad (8)$$

The impression words in the same category are equally weighted by the corresponding weights of the category given by \mathbf{v} .

This is metadata due to weighted impression word categories, which is output by the media-lexicon transformation operator for music data. We can produce a set of weighted impression words from a music media data given in the form of MIDI.

3.3 Construction of Operator to Generate Facial Expression

A set of construction operators to generate facial expression for each fundamental feeling (basic emotion) such as happiness, surprise, fear, anger, disgust, sadness is based on Facial Action Coding

Table 1: A part of Action Unit.

AU	Description
<u>1</u>	Inner Brow Raiser
<u>2</u>	Outer Brow Raiser
<u>4</u>	Brow Lower
<u>5</u>	Upper Lid Raiser
<u>6</u>	Cheek Raiser
<u>7</u>	Lid Tighter
<u>9</u>	Nose Wrinkler
<u>10</u>	Upper Lip Raiser
<u>12</u>	Lip Corner Puller
<u>15</u>	Lip Corner Depressor
<u>17</u>	Chin Raiser
<u>20</u>	Lip stretcher
<u>23</u>	Lip Tighter
<u>24</u>	Lip Pressor
<u>25</u>	Lips part
<u>26</u>	Jaw Drop
<u>27</u>	Mouth Stretch

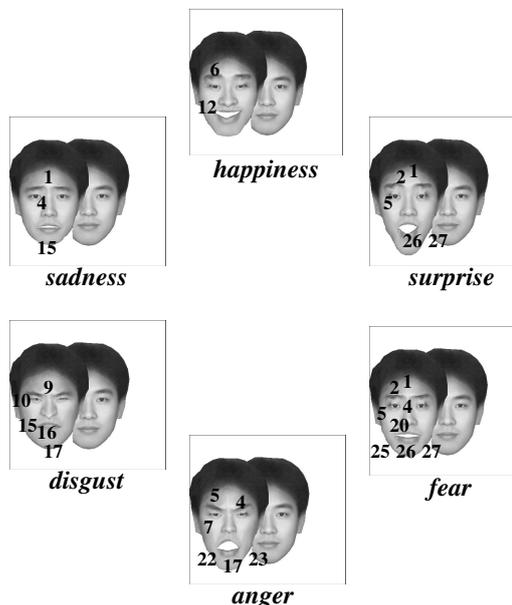


Figure 5: Relations between basic emotions and AU

System(FACS)(P.Ekman et.al 1978, Ekman & Friesen 1987).

3.3.1 Facial Action Coding System

Facial Action Coding System (FACS) by research of P. Ekman and W.V. Friesen can be used for showing a motion of a face based on dissection analysis of action of a face. In an objective facial expression consultation system, this is one of the methods currently used strictly and most widely.

P. Ekman and W.V. Friesen have shown how the appearance of a face changes with expansion and contraction of the each part of a face. And, they have clarified the method of determining how each emotion relates to each part of a face.

FACS describes facial expression with the combination of some Action Units (AU's). AU's are the minimum units of facial expression operations which are visually discernible. Table 1 shows a part of explanation of AU identifiers and their operations.

Moreover, the research results(P.Ekman et.al 1978, Ekman & Friesen 1987) have shown some combination of AU's for expressing each basic emotion. Figure 5 shows typical combination. The numbers in each face in Figure 5 express the identifiers of AU's, and the face in the back expresses the expressionless face with which feeling is not expressed. These facial expression are constructed using "Face Tool" (FaceTool).

The basic emotions are further arranged in a circle so that emotions are adjacent to other emotions to which they have similarities. The combination of these basic emotions can express a complicated facial expression.

3.3.2 An Implementation Method for Construction of Facial Expression

This section shows an implementation method for construction of facial expression. This operator consists of three steps:

Step 1 : Composition of Transformation Matrix F .

Transformation Matrix F is composed by the correlations between each basic emotion and Action Units (AU's) which are the minimum units of facial expression. The correlations are

presented in (P.Ekman et.al 1978, Ekman & Friesen 1987) and shown in Figure 5.

Step 2 : Composition of a basic emotion vector \mathbf{w} .

We form a basic emotion vector \mathbf{w} which has the weights of the basic emotion, and the vector is defined as

$$\mathbf{w} = (w_1, w_2, \dots, w_6)^T. \quad (9)$$

Step 3 :Construction of facial expression

Transformation Matrix F transforms a basic emotion vector \mathbf{w} to the weights \mathbf{u} of the AU's.

$$\mathbf{u} = F\mathbf{w} \quad (10)$$

It is possible to construct an expression corresponding to each basic emotion by reflecting the weights \mathbf{u} . We can construct facial expression from basic emotions.

4 Visualization of Music Impression in Facial Expression to Represent Emotion

This section shows a visualization method of music impression in facial expression to represent emotion. This method can measure the relationship between each impression of music data and face expression. In section 4.1, we represent an associative heterogeneous mediadata search method. In section 4.2, we propose a visualization method of music impression in facial expression to represent emotion.

4.1 An Associative Heterogeneous Mediadata Search Method

An associative heterogeneous mediadata search method is shown in Figure 6.

The media-lexicon transformation operator is applied to extract impression words from each mediadata. The mathematical model of meaning(Kitagawa & Kiyoki 1993, Kiyoki, Kitagawa & Hayama 1994) is applied to extract semantically related each word. Therefore this model can measure the relation of each word extracted from media-lexicon transformation operator for each mediadata. By these functions,

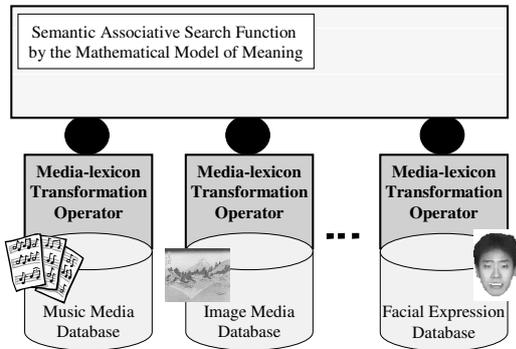


Figure 6: A fundamental framework for an associative heterogeneous mediadata search method.

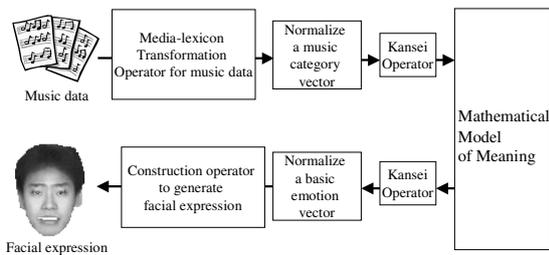


Figure 7: The process of a visualization method of music impression in facial expression to represent emotion.

this method can measure the relationship between each impression of heterogeneous mediadata.

By realization of this method, an integration appropriately corresponding to the impression between heterogeneous mediadata on meta-level are realized easily. This method can generate new information by the integration appropriately corresponding to the impression between heterogeneous mediadata on meta-level. This method realizes to bridge over heterogeneous mediadata which exist independently as different database resources.

4.2 An Visualization Method of Music Impression in Facial Expression to Represent Emotion

In this section, we show a visualization method of music impression in facial expression to represent emotion. This method can composite facial expression corresponding to impressions of a music data.

Facial expression is one of nonverbal behaviors. The facial expression is important as media which convey various emotion effectively. Assuming that an integration between existing mediadata and facial expression is realized, the interface corresponding to human Kansei is realized.

The process of this method is shown Figure 7.

This method consists of following operations.

1. Mathematical model of meaning
The Mathematical model of meaning can measure the semantic relation between each word. This function measures correlations between 6 basic emotion words and 8 impression word categories with weights from the media-lexicon transformation operator for music data.

This model has shown in section 2.

2. Media-lexicon transformation operator for music data

The media-lexicon transformation operator for music data can extract the weighted impression word categories corresponding to the impression of the music data.

This function has shown in section 3.2.

3. Normalize a music category vector
Impression word categories corresponding to the music data and their weights are output by the media-lexicon transformation operator. However, these weights generally have not been normalized.

The following formulas f_N are applied in this paper.

$$f_N(v_{c_1}, v_{c_2}, \dots, v_{c_8}) : (v'_{c_1}, v'_{c_2}, \dots, v'_{c_8}) \\ \mapsto \left(\frac{v_{c_1}}{\max_1}, \frac{v_{c_2}}{\max_2}, \dots, \frac{v_{c_8}}{\max_8} \right). \quad (11)$$

$\max_1, \max_2, \dots, \max_8$ denote the maximum weight values of each words category. Details are presented in reference (Kitagawa, Nakanishi & Kiyoki 2004).

4. Construction operator to generate facial expression

The construction operator to generate facial expression can construct facial expression from the basic emotion vector which is correlation values measured in the mathematical model of meaning.

This function has shown in section 3.3.

5. Normalize a basic emotion vector
The basic emotion vector which consists of 6 correlations of basic emotions for construction of facial expression is extracted by measurement of the relation between basic emotion words and impression words of music data in the mathematical model of a meaning. However, the basic emotion vector generally has not been normalized. The following formulas are applied in this paper.

The basic emotion vector \mathbf{fev} which consists of 6 non-normalization correlations extracted by the mathematical model of meaning is defined as

$$\mathbf{fev} = (b_1, b_2, \dots, b_6)^T. \quad (12)$$

This vector is normalized as follows:

$$\mathbf{fev}' = (b'_1, b'_2, \dots, b'_6)^T. \quad (13) \\ b'_i = \frac{b_i}{\sum_{j=1}^6 b_j}.$$

It is shown in reference (Ekman & Friesen 1987) that a complicated facial expression can be constructed with the combination of 6 basic emotions. Each AU is independent anatomically. These formulas are normalization to the value showing the rate which each basic emotion combine.

Moreover, there is a risk that the small amount of features may generally be impurities which worsen results. The feature of facial expression is more clarified by removing these values. These are shown as follows, using the removed value as w_i .

$$w_i = \begin{cases} b'_i & (b'_i \geq \varepsilon) \\ 0 & (b'_i < \varepsilon) \end{cases} \quad (14)$$

$$\varepsilon = \frac{\sum_{j=1}^6 b'_j}{6}.$$

Thus the normalized basic emotion vector is constituted.

$$\mathbf{w} = (w_1, w_2, \dots, w_6)^T. \quad (15)$$

These formulas are not always determined in this normalization method, because there is a limit in finding suitable normalization formulas in an experiment. These formulas as the normalization method are open to discussion. These normalization formulas need verification by the specialist. This verification is a future work.

6. Kansei operator

The Kansei operator (Kitagawa, Nakanishi & Kiyoki 2004) is used to adjust the expression with the interpretation of human sensitivity computed by the logarithmic function based on Fechner's law (Ohyama et al. ed. 1994). This function and a semantic associative search method make it possible to realize semantic search according to the human Kansei for multimedia data.

When impression word weights are composed for each feature, Kansei operator positions the sum total of each of the features as the stimulus strength and uses Fechner's law to obtain the sensation magnitude corresponding to that stimulus as the composed weight.

(a) Fechner's law

E.H. Weber has shown that human beings perceive the ratio of the difference in the magnitudes of objects rather than perceives the difference between the magnitudes of objects by the discrimination experiment of weights. Fechner named this fact, which Weber had discovered, Weber's law.

Fechner supposed that Weber's law is generally applied and leads to

$$d\gamma = k \frac{d\beta}{\beta}, \quad (16)$$

where k : proportionality constant; β : magnitude of a stimulus (hereafter stimulus strength); γ : sensation magnitude; $d\beta, d\gamma$: infinitesimal increases in the stimulus strength and sensation magnitude.

By the integration of (16),

$$\gamma = k(\log \beta - \log b), \quad (17)$$

where $\log b$ is the integration constant. Therefore, γ is as follows:

$$\gamma = k \log \frac{\beta}{b}. \quad (18)$$

The sensation magnitude is proportional to the logarithm of the stimulus strength. This is called the Fechner's law.

(b) Construction of the Kansei operator

Each feature assigned to each impression word can be viewed as a stimulus in that feature. Obtaining the sum totals of each feature can be thought of as obtaining the stimulus strength in each feature possessed by the mediadata. Therefore, the sum total of each feature in each impression word can be assigned the meaning of the stimulus strength of that feature.

Kansei operator g is shown as follows;

$$\mathbf{y} = (y_1, y_2, \dots, y_n)^T,$$

$$g(\mathbf{y}) := (\gamma_1, \gamma_2, \dots, \gamma_n)^T, \text{ and}$$

$$\gamma_j = \begin{cases} k \log_\alpha |y_j| + 1 & (y_j > 0) \\ 0 & (y_j = 0) \\ -(k \log_\alpha |y_j| + 1) & (y_j < 0) \end{cases} \quad (19)$$

where k and α are the parameters which can be set up as sensational volumes. These parameters are taken as $k = 1$, $\alpha = 6$ in the case of music data (Kitagawa, Nakanishi & Kiyoki 2004), and $k = 10$, $\alpha = 18$ in the case of facial expression by our pilot studies.

5 Experiments

To verify the effectiveness of this method, we built an experimental system based on this method, and performed verification experiments.

5.1 Experimental environment

To create metadata space MDS , we used the English-English dictionary *Longman Dictionary of Contemporary English* (Summers et al. ed. 1987). This dictionary uses only approximately 2,000 basic words to explain approximately 56,000 headwords. We created the data matrix M in subsection 1 of section 2 by treating basic words as features and setting the element corresponding to a basic word to "1" when the basic word explaining a headword had been used for an affirmative meaning, setting it to "-1" when the basic word had been used for a negative meaning, setting it to "0" when the basic word was not used, and setting it to "1" when the headword itself was a basic word. In this way, we generated the metadata space MDS , which is an orthonormal space of approximately 2000 dimensions. This space can express 2^{2000} different phases of the meaning.

The facial expression construction part in this experiment system is realized by utilizing "Face Tool" (FaceTool).

5.2 Experimental Method

We verify output results which are the generated facial expression by the some music data in this experimental system.

We use 4 well-known pieces which have rather apparent impression to anyone in MIDI format for some input data in this system. These pieces are "Clap your hands", "Brahms 3rd Symphony", "Song of four seasons", and "Silent night holy night". The well-approved impressions of the pieces are as follows: "Clap your hands", which goes like clap your hands if you are happy, has impression of merry, happy, and joy. "Brahms 3rd Symphony" has impression of heavy. "Song of four seasons", which goes like one who loves spring has pure heart, has impression of tender, sad and sentimental. "Silent night holy

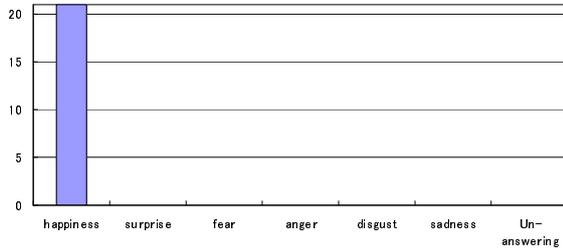


Figure 8: The result of subject investigation about impression of “Clap your hands”.

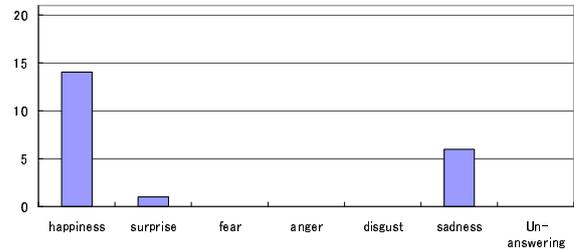


Figure 11: The result of subject investigation about impression of “Silent night holy night”.

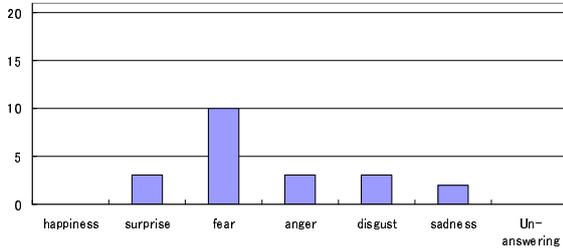


Figure 9: The result of subject investigation about impression of “Brahms 3rd Symphony”.

night” has impression of holy and solemn. This experiment system inputs these music data, and 4 facial expressions corresponding to those impressions are extracted in this experiment. We conduct hearing investigations about impressions of these facial expressions and impressions of these music data.

5.3 Experiment Results

First, hearing investigation was conducted about impressions of these music data. Subjects of 21 adult men and women are asked to select the appropriate impression corresponding to music data in 6 items. The items are “happiness”, “surprise”, “fear”, “anger”, “disgust”, and “sadness”. These items are the same as basic emotions for facial expression.

The results of these investigations about each impression about “Clap your hands”, “Brahms 3rd Symphony”, “Song of four seasons”, and “Silent night holy night” are shown in Figure 8, 9, 10, and 11.

In case of “Clap your hands” shown in Figure 8, all subjects have answered “happiness”. This result corresponds to the impression of this music that we assumed.

In case of “Brahms 3rd Symphony” shown in Figure 9, more than 40% subjects have answered “fear”. However other subjects have answered “surprise”, “anger”, “disgust”, and “sadness”. We assumed the impression of this music to be heavy. The impression of “heavy” is close on sadness, angry, and fear in meaning. Thus this result corresponds to the impression that we assumed.

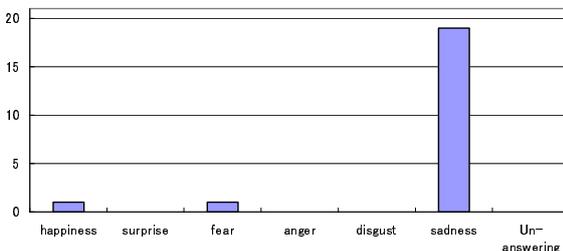


Figure 10: The result of subject investigation about impression of “Song of four seasons”.

Table 2: The result extracted from “Clap your hands”.

C6	59.019306
C5	37.261279
C7	13.467024
C8	-6.346211
C4	-11.518259
C3	-17.185234
C1	-25.459551
C2	-34.546004

In case of “ Song of four seasons” shown in Figure 10, more than 90% subjects have answered “sadness”. This result corresponds to the impression of this music that we assumed.

In case of “Silent night holy night” shown in Figure 11, more than 60% subjects have answered “happiness”, and about 30% subjects have answered “sadness”. This result never corresponds to the impression of this music that we assumed.

The results extracted by the media-lexicon transformation operator for music shown in section 3.2 from 4 MIDI data are shown in the Table 2, 3, 4, and 5.

In case of “Clap your hands” shown in Table 2, weight of the impression word category of C6 expressing “happy” is the largest. This result corresponds to the subject investigation.

In case of “Brahms 3rd Symphony” shown in Table 3, weights of the impression word categories of C2 and C3 expressing “sad”, “heavy”, and “longing” are large. The impression of “heavy” is close on sadness, angry, and fear in meaning. This result almost corresponds to the subject investigation.

In case of “ Song of four seasons” shown in Table 4, weights of the impression word categories of C2 and C3 expressing “sad”, “heavy”, and “longing” are large. This result corresponds to the subject investigation.

In case of “Silent night holy night” shown in Table 5, weight of the impression word category of C1 expressing “serious” is the largest. However, weight of the impression word category of C6 expressing “happy” is also large. This song has not only serious but also happy. The impression word category of C2

Table 3: The result extracted from “ Brahms 3rd Symphony ”.

C2	21.727009
C3	14.127015
C7	6.552777
C8	-4.168117
C4	-8.970285
C6	-18.778780
C5	-19.791271
C1	-19.205445

Table 4: The result extracted from “ Song of four seasons”.

C3	17.863154
C2	14.470195
C4	1.070744
C7	-2.208193
C8	-7.584915
C5	-9.435428
C6	-9.925302
C1	-17.619686

Table 5: The result extracted from “ Silent night holy night”.

C1	21.469676
C6	11.058502
C5	9.959276
C4	6.508946
C8	5.181874
C7	-6.144448
C2	-9.816415
C3	-12.011969

expressing “sadness” which about 30% subjects have answered has negative weights. This result never corresponds to the subject investigation. We find that it is difficult for such song to decide impression.

The results of measurement of correlations between 6 basic emotion words and 8 impression word categories with weights utilizing the mathematical model of meaning are shown in the Table 6, 7, 8, and 9.

In case of “Clap your hands” shown in Table 6, correlation of “happiness” is the largest in 6 basic emotion words. “Happiness” is close to “C6” semantically. In case of “Brahms 3rd Symphony” shown in Table 7, correlation of “sadness” is the largest in 6 basic emotion words. “Sadness” is close to “C2” or “C3” semantically. In case of “ Song of four seasons” shown in Table 8, correlation of “sadness” is also the largest in 6 basic emotion words. In case of “Silent night holy night” shown in Table 9, correlations of “happiness”, “surprise” and “anger” are large in 6 basic emotion words. “C1” is close to “anger” and “surprise” in the mathematical model of meaning utilizing the space constructed by Longman Dictionary of Contemporary English(Summers et al. ed. 1987). “C1” is closer to “C8” expressing “emphatic” than “C4” expressing “calm” in Hevner’s work. Actually, “anger” and “surprise” are emphatic expressions. These results are appropriately measured by the mathematical model of meaning.

Finally, the experimental results which facial expression are constructed are shown in Figure 12, 13, 14, and 15.

In the case of Figure 12, “happiness” as facial expressions are automatically created. In the case of Figure 13, facial expression which mixed “sadness” and “fear” is automatically created. In the case of Figure 14, “sadness” as facial expressions are automatically created. In the case of Figure 15, facial

Table 6: The result of measurement correlations (“Clap your hands”).

happiness	0.153885
anger	0.134517
surprise	0.131953
fear	0.127711
disgust	0.105515
sadness	0.077445

Table 7: The result of measurement correlations (“Brahms 3rd Symphony”).

sadness	0.340237
anger	0.177366
fear	0.173911
surprise	0.143764
disgust	0.123588
happiness	0.097354

Table 8: The result of measurement correlations (“Song of four seasons”).

sadness	0.326680
anger	0.214740
fear	0.204385
disgust	0.190738
surprise	0.183363
happiness	0.139167

expression which mixed “happiness” and “surprise” are automatically created.

Moreover, the results by subjects of 21 adult men and women are shown Figure 16, 17, 18, and 19. These results show whether the output result corresponds to the impression of input pieces. All subjects are asked to select the most appropriate item from “Exaggerated”, “Correct”, “Almost Correct”, “Comfortable”, “Slightly Different”, and “Totally Different”.

In the case of “Clap your hands” shown in Figure 16, the more than 90% subjects have answered “Correct”, “Almost Correct” and “Comfortable.” This result is shown that this facial expression corresponds to impressions of this song.

In the case of “Brahms 3rd Symphony” shown in Figure 17, the more than 70% subjects have answered “Correct”, “Almost Correct” and “Comfortable.” This result is shown that this facial expression corresponds to impressions of this song.

In the case of “Song of four seasons” shown in Figure 18, the more than 80% subjects have answered “Correct”, “Almost Correct” and “Comfortable.” This result is shown that this facial expression corresponds to impressions of this song.

In the case of “Silent night holy night” shown in Figure 18, the more than 70% subjects have answered “Exaggerated”, “Slightly Different” and “Totally Different”.

The media-lexicon transformation operator for music extracts not only “C1” expressing “serious” but also “C6” expressing “happy” from “Silent night holy night”. Hereby, facial expression which mixed “happiness” and “surprise” are automatically created. These results are appropriate for the experimental system. In contrast, “Silent night holy night” has the more than 60% subjects who have answered “happiness” like “Clap your hands”. However, the impression of “Silent night holy night” is holy and solemn unlike “Clap your hands” certainly. In the case of this song, it is thought that the impression of other elements such as lyrics, a title and so on

Table 9: The result of measurement correlations (“Silent night holy night”).

happiness	0.228688
surprise	0.185209
disgust	0.178705
anger	0.176306
fear	0.175334
sadness	0.114702

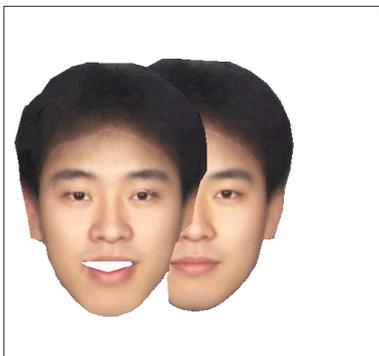


Figure 12: The result (“Clap your hands”).

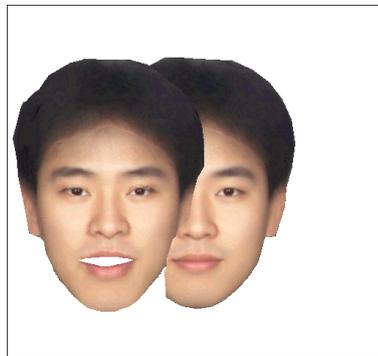


Figure 15: The result (“Silent night holy night”).

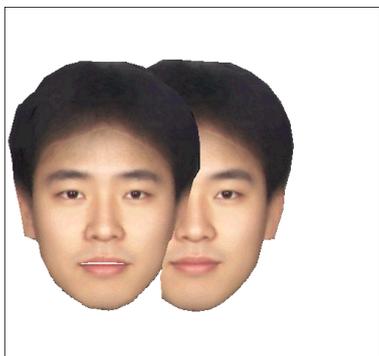


Figure 13: The result (“Brahms 3rd Symphony”).

is large. Assuming that a media-lexicon transformation operator for other elements is realized, clearer impression words will be extracted and appropriate facial expression will be constructed. A realization of the media-lexicon transformation operator for other elements is our future work.

As shown in those results, we have clarified that our method constructs various facial expressions from arbitrary music data.

6 Conclusion

In this paper, we proposed a visualization method of music impression in facial expression to represent emotion. We clarified the effectiveness of this method by showing several experiment results.

This method realizes an integration corresponding to the impression between existing mediadata and nonverbal behaviors that convey various emotions effectively. The interface corresponding to human Kansei with less difficulty for a user is realized by this method which realize a integration appropriately cor-

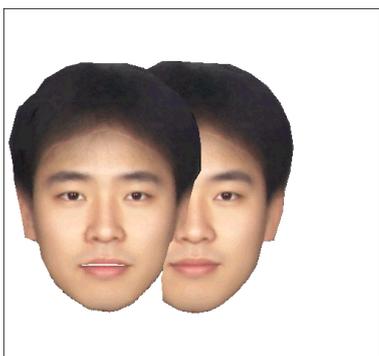


Figure 14: The result (“Song of four seasons”).

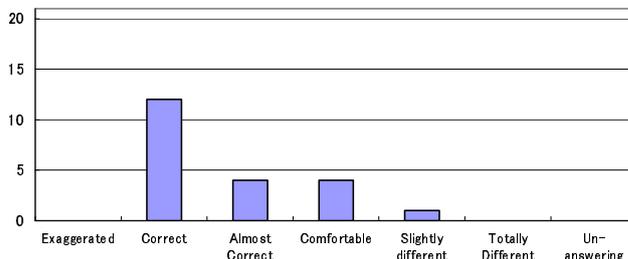


Figure 16: The result of subject investigation about “Clap your hands”.

responding to the impression between existing mediadata and facial expression.

We believe that this method which realizes an integration appropriately corresponding to the impression between existing mediadata and facial expression can be used for a realization of the interface corresponding to human Kansei with less difficulty for a user.

As our future work, we will realize a learning mechanism according to individual variation. We will also consider analytical evaluation and verification by the specialist for facial studies. Furthermore, we will apply this method to various type of existing mediadata and nonverbal behaviors.

Acknowledgment

We really appreciate Prof. Yasushi Kiyoki, Faculty of Environmental Information, Keio University, for his substantial supports to enhance the presentation of the paper.

References

Ekman,P. & Friesen,W.V. (1978), Facial Action Coding System, *Consulting Psychologist Press*.

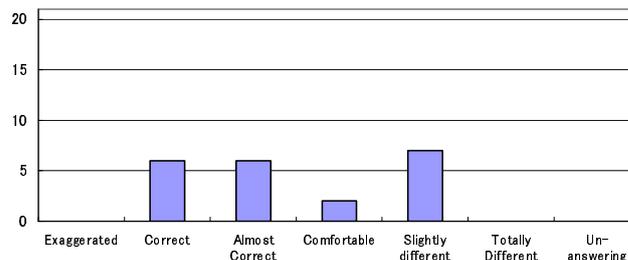


Figure 17: The result of subject investigation about “Brahms 3rd Symphony”.

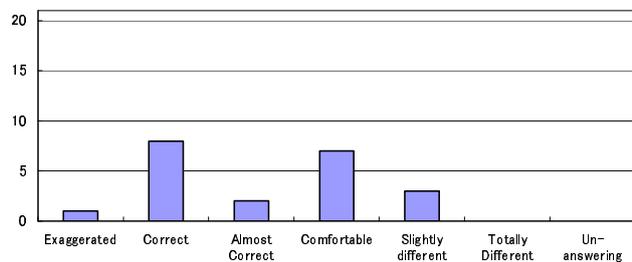


Figure 18: The result of subject investigation about “Song of four seasons”.

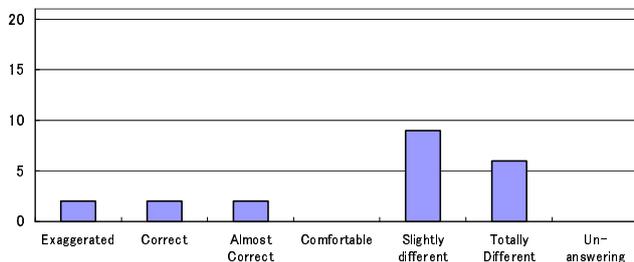


Figure 19: The result of subject investigation about “Silent night holy night”.

Ekman,P. & Friesen,W.V., Kudo,T.(translator and editor) (1987), A guide to expression analysis-The meaning hidden in expression is explored, *Sisin-Shobou Press*.

Choi,CS., Harashima,H. & Takebe,T. (1990), 3-dimensional facial model-based description and synthesis of facial expressions, *The Transactions of the Institute of Electronics, Information and Communication Engineers*, Vol.J73-A, No.7, pp.1270-1280.

Hara,F. & Kobayashi,H. (1996), Real-time facial interaction between human and 3D face agent, *Proc.5th IEEE International Workshop on Robot and Human Communication (RO-MAN'96)*, pp.401-409.

Kitagawa,T. & Kiyoki,Y. (1993), The mathematical model of meaning and its application to multi-database systems, *Proceedings of 3rd IEEE International Workshop on Research Issues on Data Engineering: Interoperability in Multidatabase Systems*, pp.130-135.

Kiyoki,Y., Kitagawa,T. & Hayama,T. (1994), A metadatabase system for semantic image search by a mathematical model of meaning, *ACM SIG-MOD Record*, vol. 23, no. 4, pp.34-41.

Kitagawa,T. & Kiyoki,Y. (2001), Fundamental framework for media data retrieval system using media lexico transformation operator, *Information Modelling and Knowledge Bases*, vol.12, pp. 316-326.

Kitagawa,T., Nakanishi,T. & Kiyoki,Y. (2004), An Implementation Method of Automatic Metadata Extraction Method for Music Data and its Application to a Semantic Associative Search, *Systems and Computers in Japan*, Vol.35, No.6, pp59-78.

Hevner,K. (1935), Expression in music: A discussion of experimental studies and theories, *Psychological Review*, Vol. 42, pp. 186-204.

Hevner,K. (1936), Experimental studies of the elements of expression im music, *American Journal of Psychology*, Vol. 48, pp. 246-268.

Hevner,K. (1937), ‘The affective value of pitch and tempo in music, *American Journal of Psychology*, Vol. 49, pp. 621-630.

Umemoto,T.(editor). (1966), Music Psychology, *Seishin-Shobo Press*.

FaceTool
<http://www.hc.t.u-tokyo.ac.jp/project/face/>

Oyama,T., Imai,S. & Wake,T.(editors) (1994), New edition, Handbook of sensory and perceptive psychology, *Seishinshobou Press*.

Sumners,D. et al. (1987), Longman dictionary of contemporary English, *longman*.

Modelling Human Perception to Leverage the Reuse of Concepts across the Multi-sensory Design Space

Keith V. Nesbitt

School of Information Technology
Charles Sturt University
Panorama Av, Bathurst, NSW
knesbitt@csu.edu.au

Abstract

Information Visualisation is an emerging discipline that concerns the design of interactive computer systems that provide the user with a visual model of abstract data. Information Visualisation implies a mapping from the data attributes to the units of visual perception. Information Sonification is an embryonic field that uses sound rather than imagery to present abstract data. Information Sonification, implies a mapping from the data attributes to the units of auditory perception. In both these fields the need to describe appropriate mappings between the data and the units of perception has led to models or taxonomies that describe the available design space. While these models of the visual design space and the auditory design space may be appropriate for people working in a single sensory domain, these models based purely on sensory attributes are very disjoint. However, for designers who wish to consider a multi-sensory solution to information display, these disjoint models of the different sensory domains make it difficult to compare and contrast the possible mapping choices.

This paper describes existing conceptual models of the visual and auditory design space and then proposes a different conceptual modelling of the multi-sensory design space. This new model describes the units of perception but is not based on sensory attributes, but typical information metaphors. Throughout the paper all discussions are illustrated using the UML modelling notation which is a standard notation used to document the design of software systems.

Keywords: Perception, Modelling, Multi-sensory

1 Introduction

The idea of mining large abstract data sets for useful patterns is an attractive proposition, especially at a time when most companies are growing larger and larger stocks of data. The most traditional notion of data mining

is an automated process, which involves running rule-finding algorithms across the data to automatically detect patterns. However, there is also a growing interest in the idea of developing tools that support human pattern recognition within large data sets. Such human perceptual tools present the data to the user's senses (vision, hearing, touch) in a way that the user can search for useful patterns.

It might be expected that Human Perceptual Tools are particularly useful where; unpredictable exceptions may occur in the data; heuristics are required to filter subtle variations; the target is unknown or cannot be precisely formalised by rules and; the problem requires intuitive knowledge that is hard to formalise, such as, past experience.

During the 1990s, the accent for Human Perceptual Tools was on designing visual displays of data. This approach is sometimes called *visual data mining* (Soukup 2002), although the more general term is *information visualisation* (Card, Mackinlay et al. 1999). A number of example applications have been described and the field is beginning to develop a more theoretical basis (Card, Mackinlay et al. 1999).

By contrast the use of other senses, such as, hearing and haptics (touch) to display abstract data are fairly embryonic. The term *information sonification* is used to describe auditory models of abstract data and despite a number of validated uses of sound for finding patterns in abstract data (Kramer, G. 1994) the field can probably be best described as immature.

Haptic displays are still relatively uncommon although some novel applications have been developed, for example, the use of haptic displays for investigating patterns within force fields (Brooks, Ouh-Young et al. 1990) and fluid flow models (Nesbitt, Gallimore et al 2001).

Regardless of the particular sensory modality, the designer of a human perceptual tool can describe the design as a mapping between the data and the some characteristics of the model. A common approach is to map the data to the display artefacts that can be perceived by the user. Assuming there are no shortcomings in the display itself, the possible range of artefacts can be described as the fundamental elements or units of human perception. For example *colour* and *shape* are two of the visual perceptual units available to a designer. If the

Copyright © 2006, Australian Computer Society, Inc. This paper appeared at the Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 53. Markus Stumptner, Sven Hartmann, and Yasushi Kiyoki, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

display shows stock market data then price might be mapped to *colour* and the trade volume to *shape*.

Good designers must understand the range of possibilities and therefore one of first steps in formalising the design process is to categorise the design space. Some rigorous attempts to categorise visual artefacts (Bertin 1981, Card and Mackinlay 1997) have resulted in descriptions of the visual design space (figure 2). Within the auditory design space some common methodologies, such as earcons (Blattner, D. Sumikawa, et al 1989) and auditory icons (Gaver 1986) have developed. There is also some agreement about the perceived characteristics of sound (Kramer, G. 1994) which can be used for sonification, although perceptual qualities, such as, timbre have proved difficult to synthesize in a controlled way.

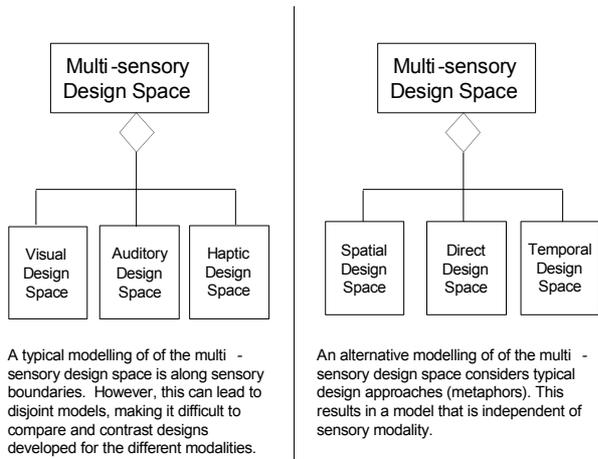


Figure 1: High-level divisions of the design space

One problem with these existing descriptions of the visual and auditory design space is that they do not enable the reuse of concepts across the different senses. Of course these models were never intended for multi-sensory display and so it is not surprising that there is a lack of reusable concepts which can be used to classify the multi-sensory design space.

Rather than use sensory divisions to model the perceptual design space, the approach taken in this work is to base the model on general information design strategies or metaphors. This will lead to a conceptual model that is not specific to any single sense, and allows the designer to reuse concepts across sensory domains.

2 Previous Models of the Design Space

The size of the multi-sensory design space and the complex nature of human perception has led to fragmented expertise as researchers tend to narrow the scope of their work and focus on designing displays for a single sense. Therefore, a very natural division of the design space along sensory modalities occurs (figure 1). While divisions along sensory boundaries has proved useful to narrow scope by segmenting the research into haptic, visual and auditory display it has also meant that a language common to all senses has not been developed.

The fact that models are disjoint is not surprising as conceptual models of the design space for each sense have developed independently of each other and without

regard for how models from the other sensory domains might be related. For example, models of the visual design space (figure 2) and auditory design space (figure 3, 4) have little in the way of shared concepts to connect them. For example a key concept in the visual design space is "marks" which does not generalise well to auditory display (Card and Mackinlay, 1997). While "space" is very important in visual design, it is rarely discussed as an auditory design strategy.

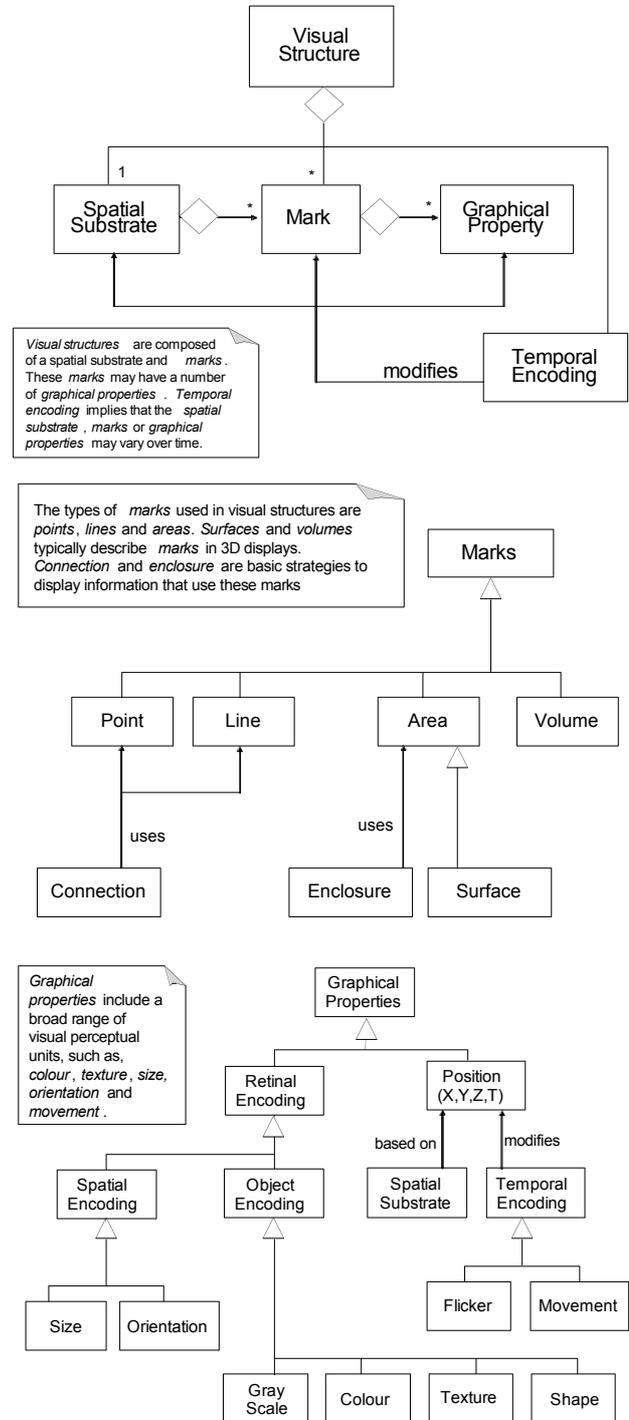


Figure 2: A model of the visual design space as proposed by Card and Mackinlay (1997) and incorporating the graphical properties described by Bertin (1981)

For researchers only interested in a single modality of display these models may well serve their needs. However, designers of multi-sensory displays must try and reconcile these disparate models and without a common language it is difficult to move between sensory domains. It is also difficult to acquire knowledge in a new sensory domain by building on previous experience with a different sensory domain. For example, experts in visualisation will find it difficult to transfer that knowledge to the sonification or haptic domain.

Lack of a common framework also makes direct comparisons between haptic, visual and auditory displays difficult. A simple example of this is when different types of data are used on the displays. This can bias the user's performance to the display which displays the data most relevant to the tasks being measured. Even where the same data is displayed, a comparison between a well-designed visual display and poorly-designed auditory display is not particularly useful. It would be nice to have a more common description of display mappings, so that designers could better compare display performance across the senses and, if required, interchange appropriate mappings between the senses.

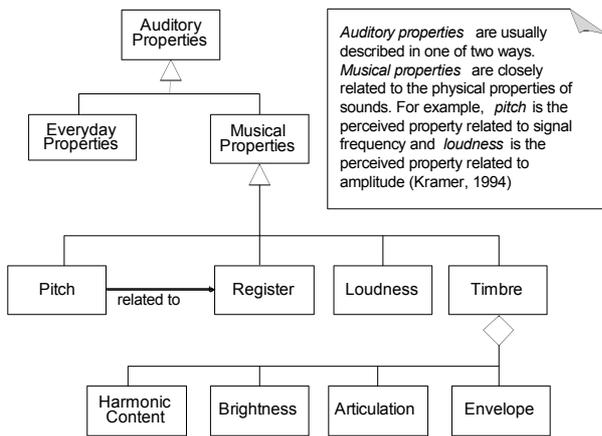


Figure 3: Descriptions of the auditory design space are usually based on musical properties (Kramer 1994)

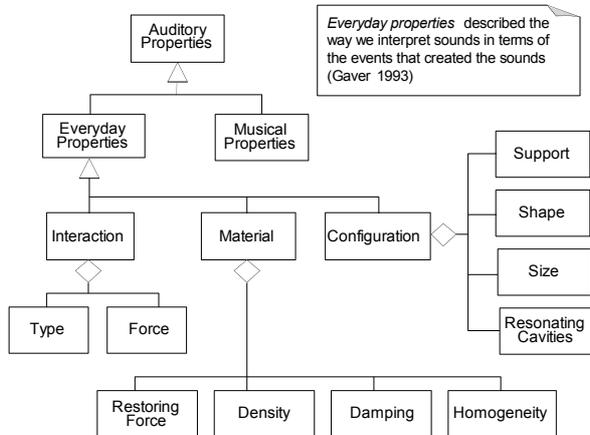


Figure 4: An alternative model of the auditory design space focuses on the way we interpret sounds in terms of the events that caused them (Gaver 1993) Gaver refers to this as everyday listening.

3 An Alternative Model of the Design Space

The MS-Taxonomy is an alternative model of the perceptual design space that divides the design space by abstracting the typical types of metaphors that have been used to design mappings between data attributes and sensory properties. The metaphors form three main classes, *Spatial Metaphors*, *Direct Metaphors* and *Temporal Metaphors*. These classes are general for all senses. The division of the design space by senses is not lost but rather forms a second, weaker division of the design space (figure 5). In software engineering terms the traditional model of the multi-sensory design space uses the concepts of *visual*, *auditory* and *haptic* for the most general base classes (figure 6). The MS-Taxonomy however uses *Spatial Metaphors*, *Direct Metaphors* and *Temporal Metaphors* as the most general base classes.

Spatial Metaphors relate to the scale of objects in space, the location of objects in space and the structure of objects in space. The key aspect of spatial metaphors is that they involve some perception of properties that depend on space. For example, *Spatial Metaphors* concern the way pictures, sounds and forces are organised in space and can be described for the visual, auditory and haptic senses. Spatial metaphors involve the perception of a quality (space) that is not associated with any particular sense. Although different classes of spatial metaphors (visual, auditory and haptic) can be described, the concepts that define a spatial metaphor are general and therefore independent of the senses. It is simply the way that each sense perceives these spatial qualities that may vary.

Temporal Metaphors are concerned with how we perceive changes to pictures, sounds and forces over time. The emphasis is on displaying information by using the fluctuations that occur over time. Because there may be differences in the way we perceive temporal patterns using each sense, *Temporal Metaphors* can be considered not only generally but also for each of the senses. *Temporal metaphors*, like *Spatial Metaphors*, involve the perception of a quality (time) that is not associated with any particular sense. Though the three different classes of temporal metaphors (visual, auditory and haptic) are described, the concepts that define a temporal metaphor are general and therefore independent of the senses.

Direct metaphors are concerned with direct mappings between sensory properties and some abstract information. The key aspect of *Direct Metaphors* is that they involve some perception of properties that depend directly on the sensory receptors involved. For example, sensory properties such as a colour for vision, pitch for hearing or surface hardness for the haptic sense. Once again, a class of *Direct Metaphors* can be defined for each sense. Unlike *Spatial and Temporal Metaphors*, *Direct Metaphors* are highly specific for each modality. Each sense perceives distinct sensory properties that are independent of space and time and directly related to the sensory receptors involved. These sensory properties can be used to display data and such mappings are described as *Direct Metaphors*. At a low level the concepts of *Direct Metaphors* are specific to each sense, however, the more general concept of a *Direct Metaphor* applies across

The display space is the region where the data is presented. All spatial metaphors have as their basis an underlying display space that is used to arrange the display elements. For example, the scatterplot defines a 2D orthogonal display space by mapping data attributes to the x and y axis. Points are then interpreted in terms of this display space. In the real world, space is perceived as constant, however in an abstract world the properties that define the space can also be designed. For example, one axis of the scatterplot could be defined as a logarithmic space. This would change the way the user interprets the relationships between point positions.

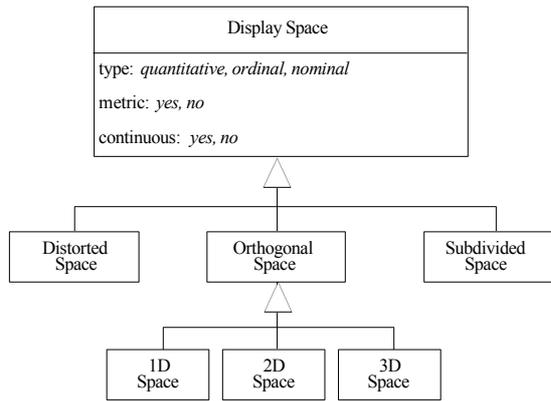


Figure 8 The types of display space

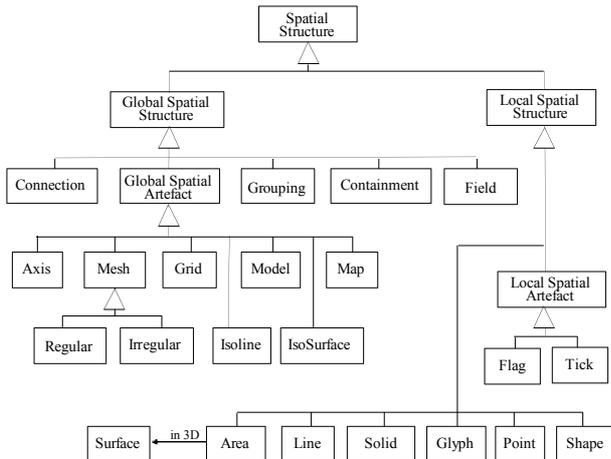


Figure 9. The types of spatial structure.

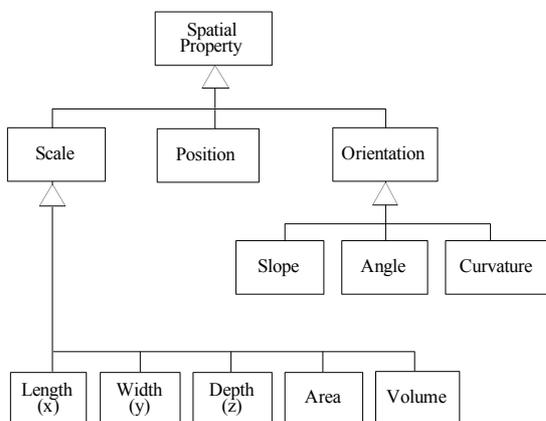


Figure 10. The types of spatial properties.

There are a number of strategies for designing the display space when presenting information and these include using orthogonal spaces (1D, 2D, 3D), distorted spaces and subdivided spaces.

In the MS-Taxonomy, the objects that occupy the display space are described as spatial structures. For example in the scatterplot, the points are spatial structures. Spatial structures also describe the arrangement of entities within the display space. For example, a group of points in the scatterplot can be considered a more global spatial structure. The MS-Taxonomy distinguishes two levels of organisation for presenting information and these are global spatial structures and local spatial structures.

Spatial structures may have spatial properties. The spatial properties used for presenting information include position, scale and orientation. Spatial properties describe qualities that are interpreted in terms of the display space. For example, in the scatterplot the position of points is used to convey information. This information is interpreted in terms of the abstract space defined by the x and y axis.

There are some points to note about spatial properties. Firstly these spatial concepts applied to the auditory sense are not as intuitive as the application of the same concepts to the visual or haptic sense. There are also a much greater number of examples of spatial metaphors to be found in the field of visualisation. This is not surprising as hearing is predominantly temporal and is more adept at identifying temporal relationships than spatial relationships (Friedes 1974). By contrast both visual and haptic perception are strongly base around interpreting space. This interpretation is supported by a distribution of cortical neurones that are organised according to the way they respond to stimuli in space (Granlund et al. 2001). Cortical auditory neurones are organised in a tonotopic way, that is, they are grouped according to how they respond to pitch (Granlund et al. 2001).

5 Modelling Direct Metaphors

In the real world a great deal of useful information is perceived directly from the properties of sights, sounds and surfaces. For example, an object may have a particular hardness or surface texture. Objects in the real world may also be recognised on the basis of visual properties such as colour or lighting or interpreted on the basis of auditory properties like pitch and timbre. Abstract information can also be interpreted in terms of these direct properties.

An important distinction between spatial metaphors and direct metaphors is that direct metaphors are interpreted independently from the perception of space. While the concepts of spatial metaphors apply generally for each sense this is not true for direct metaphors. There is very little intersection, for example; between the low level concepts of direct visual metaphors and the low level concepts of direct auditory metaphors. This is not surprising as direct metaphors relate to the properties that the individual sensory organs can detect.

Direct metaphors (figure 11) are concerned with direct mappings between the properties perceived between each sense and some abstract information. Direct metaphors consider the design concepts of *spatial structure* and *direct properties*.

Spatial structures are a component of spatial metaphors that can be used to convey information. These structures can be encoded with additional information by using a directly perceived property of any sense. For example, colour can be used with a visual display or hardness with a haptic display.

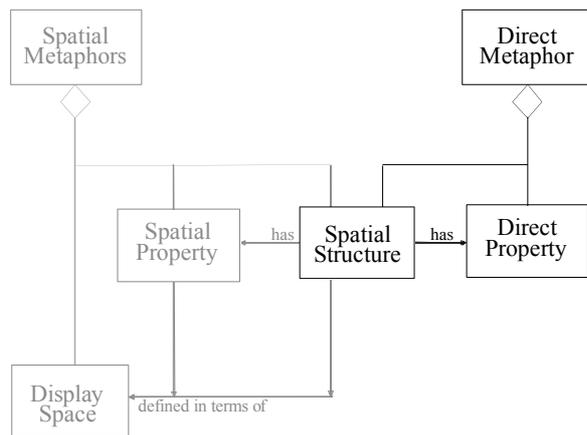


Figure 11. The general concepts that describe Direct Metaphors. These concepts are very specific to the properties of the world that each sense perceives.

The key component of direct metaphors is the direct property used to convey the information. In terms of design, the effectiveness of a direct metaphor is independent of the display space and the spatial structure. However, in some cases there needs to be consideration for the size of the spatial structure. For example, very small areas of colour may not be visible to the user, or a haptic surface may be too small for the user to feel.

The ability to accurately interpret direct properties varies between senses and properties. In general, the perception of all direct properties is of insufficient accuracy to allow accurate judgement of quantitative values (Sekuler and Blake 1990). This suggests that direct properties should only be used to encode ordinal or nominal categories of data. Because direct properties such as colour, pitch or hardness are continuous they can be mapped to continuous data. However, it should not be assumed that a user is capable of interpreting exact data values represented as direct properties.

The MS-Taxonomy distinguishes between direct visual and direct auditory metaphors. At a low-level of the hierarchy, the concepts do not abstract across the senses. This makes it difficult for direct metaphors to be directly compared between senses. For example, it makes little sense to compare the ability of the visual and auditory sense at judging the *pitch* of sounds. However, for the designer the higher level concept of a *direct property* is still relevant as it applies across all senses. Therefore at a conceptual level the designer can consider substituting one direct property with another. For example, the direct

visual property of *colour* could be substituted with the direct haptic property of *hardness* for representing categories of data.

Many of the concepts used to describe direct properties are familiar to display designers as they overlap with existing sensory-based models of the design space. Much previous work has been done in the area of direct visual properties and to a lesser extent direct auditory properties. Because haptic display is a relatively new area and involves a complex range of sensations, describing the concepts that make up direct haptic properties is difficult. Arguably the MS-Taxonomy needs some discussion and refinement centred around the low level concepts that make up direct haptic metaphors.

Direct visual metaphors use direct mappings from the attributes of data to the perceived properties of sight. These properties include colour hue, colour saturation and visual texture (figure 12).

Using direct visual properties to represent information has been well studied. Bertin described the basic properties of visual objects as *retinal properties* (Bertin 1981). Bertin's *retinal properties* include the scale and orientation of objects. These concepts are dependent on the visual space and so are included in the MS-taxonomy as visual spatial metaphors. However, Bertin's other *retinal properties* are all concepts within direct visual properties. They are:

- colour - hue
- colour - saturation
- colour - intensity (grey scale, value)
- visual texture
- direct visual shape.

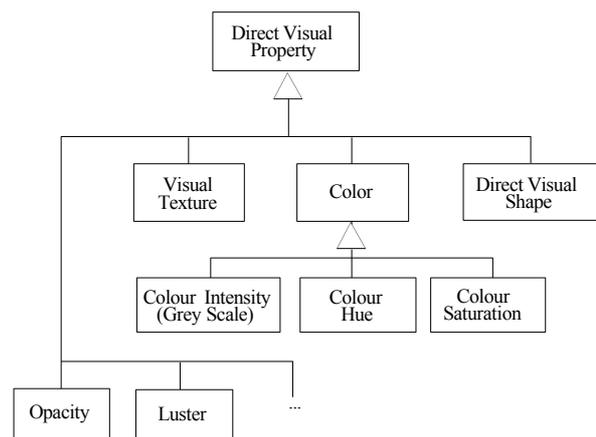


Figure 12. Direct Visual Properties

Direct auditory metaphors use direct mappings from the attributes of data to the perceived properties of sound. As mentioned, the use of direct auditory properties for representing abstract data is an embryonic field of study. Indeed many of the perceived properties of sound are not well understood (Kramer 1994) and the direct auditory properties are less generally agreed on than the visual

properties. However, the most commonly used properties of sound are:

- loudness
- pitch
- timbre.

These direct auditory properties (figure 13) have also been referred to as *musical properties* (Gaver 1994). The direct auditory properties are not independent or orthogonal. For example, the pitch of the sound affects the perceived loudness of the sound (Sekuler and Blake 1990). Furthermore, both pitch and loudness are not equally prominent to the listener (Bly 1994).

Alternative ways for defining sound properties have been developed. In particular musical listening contrasts with the concept of *everyday listening* where sound properties are interpreted in terms of the objects and events that generate the sounds (Gaver 1994). For example, the sound from a stick hitting an empty can provide information about the objects involved and the forces used to create the sound. This approach is arguably more intuitive for the user, but more difficult for the designer.

The MS-Taxonomy uses musical properties to define the design space of direct auditory metaphors. These musical properties, which are interpreted by directly listening to the qualities of the sound itself, are intuitive and simple concepts for the designer to use. Furthermore the mappings between properties and data are simple to describe.

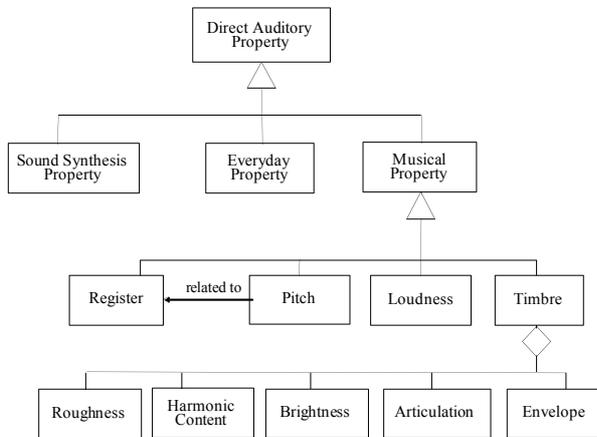


Figure 13. Direct Auditory Properties

Direct haptic metaphors use direct mappings from the attributes of data to the perceived properties of the haptic sense. These properties include surface texture, force and compliance. Figure 14 shows the different types of direct haptic properties that are principally associated with the *tactile* sense. Figure 15 shows the different types of direct haptic properties that are principally associated with the *kinaesthetic* and *force* sense. Some of the direct haptic properties, such as compliance and friction, require the combined perception of *tactile*, *kinaesthetic* and *force* stimuli. As previously noted, defining the concepts that make up direct haptic properties is somewhat rudimentary

and probably requires further consideration. The MS-Taxonomy currently uses the following direct haptic properties:

- force
- surface texture
- direct haptic shape
- compliance
- viscosity
- friction
- inertia
- weight
- vibration
- flutter.

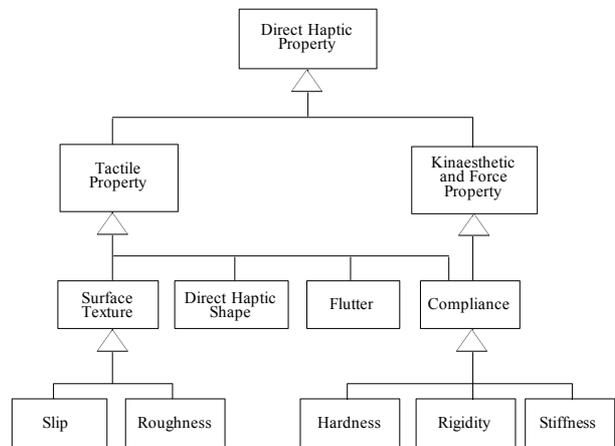


Figure 14. Direct haptic properties associated with tactile stimuli

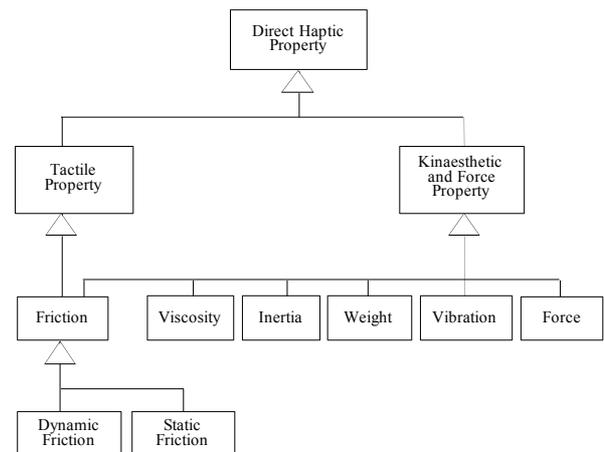


Figure 15. Direct haptic properties associated with kinaesthetic and force stimuli.

Direct metaphors map data directly to a sensory property. Although accuracy varies between direct properties, in

general, it is not possible for users to make accurate judgements about sensory properties (Sekuler and Blake 1990). Many direct properties are continuous and ordered and can be used for displaying quantitative data. However, it cannot be assumed that a user will make an accurate judgement of the value of a property. Therefore, it is more appropriate to use ordered properties for displaying ordinal data. The exceptions are those direct properties that have no ordering (colour, timbre, direct haptic shape) and these are better suited for displaying nominal data.

6 Modelling Temporal Metaphors

In the real world a great deal of useful information is dependent on the perception of time. For example, a pedestrian crossing a busy road is required to interpret the amount of time between vehicles. The rate and frequency of traffic may also impact on the pedestrian's decision of when to cross. Temporal concepts like duration, rate and frequency can also be used to encode abstract information.

Temporal metaphors relate to the way we perceive changes to pictures, sounds and haptic stimuli over time. The emphasis is on interpreting information from the changes in the display and how they occur over time. Temporal metaphors are also closely related to both spatial and direct metaphors. For example it is changes that occur to a particular spatial metaphor or direct metaphor that displays the information. (Figure 16)

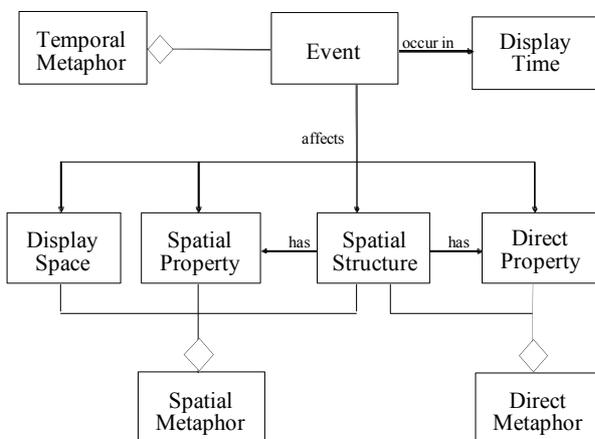


Figure 16. Temporal metaphors are dependent on the perception of time and are characterised by events that modify spatial and direct properties.

Of course all the senses require some amount of time to interpret a stimulus. This is very fast for vision, while with hearing and haptics most stimuli are more prolonged events with some temporal structure. For example, a sound stimulus is perceived by interpreting changes that occur in air pressure over time. Even a single sound event, such as a bottle breaking, contains a complex temporal pattern that is perceived over a short period of time. However, with temporal metaphors the focus is on how changes that occur in events are used to represent abstract information. That is, the focus for the designer is how temporal changes and patterns can be used to convey

information. Designing temporal metaphors is analogous in many ways to the design of music.

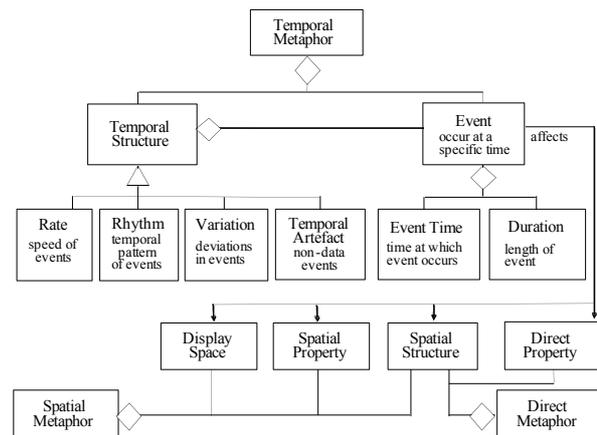


Figure 17. Temporal metaphors are often composed of a number of events that have temporal structure.

The MS-Taxonomy distinguishes between temporal visual, temporal auditory and temporal haptic metaphors. However the general concepts that describe temporal metaphors are independent of sensory modality (figure 16). It is simply the ability of each sense to perceive changes over time that need to be considered. Because the concepts abstract across the senses it is possible for temporal metaphors to be directly compared between senses. For example, the ability of the visual sense to identify a visual alarm event can be compared with the ability of hearing to identify a sound alarm or touch to identify a haptic alarm.

The design space for temporal metaphors can be described using the following general concepts:

- the display time (figure 16)
- an event (figure 16, 17, 18)
- the temporal structure (figure 17).

Temporal metaphors are composed of events that occur within the *display time*. The *display time* provides the temporal reference for the data events that are displayed. This is analogous to the way *tempo* is used in music to provide a background measure of time. The display time is not usually considered as part of the design space, but simply assumed to be constant. However, it is possible to consider the display time during the display design. For example, changing the display time could speed up or slow down the rate at which data is displayed.

Events have two main properties, the event time and the duration of the event. Both the event time and event duration are interpreted in relation to the display time. These events affect changes to the visual or auditory or display. It is these changes and the timing and duration of these changes that are interpreted by the user as information. An event can affect a change to the display space, a spatial property, the spatial structure or a direct property in the display. This allows events to be categorised by reusing many of the concepts described for

spatial metaphors and direct metaphors. The MS-Taxonomy defines the following types of event (figure 18):

- a display space event
- a movement event
- a transition event
- an alarm event.

Display space events cause a change to the perceived display space. For example, a distortion event can change the metric at a location in the display space. A navigation event can affect a change in the user's position in the display space and is usually associated with user interaction.

Movement events are related to changes in spatial properties of structures and can be characterised by properties such as direction, velocity and acceleration. Distinct types of movement events include; translation events, rotation events and scale events. Translation events involve a change to the spatial property of position. Rotation events involve a change to the spatial property of orientation. Scale events cause a change to the spatial property of scale.

The other types of events are transition events and alarm events. Transition events cause a slow change to either spatial structures or direct properties. By contrast alarm events cause a very sudden change to either spatial structures or direct properties.

A user may interpret information based on a single event. For example, a visible object changing position may be interpreted in terms of the old position and the new position, as well as the speed of movement. However, information may also be interpreted based on patterns that occur in a sequence of events. This is described as temporal structure. Types of temporal structure include the rate of events, the rhythm of events and the variations between events.

The concepts of temporal metaphors are very intuitive when described for the auditory sense. This is not surprising as hearing is usually identified as a temporal sense (Friedes 1974). Indeed many of the concepts described in temporal auditory metaphors have been developed within the field of music. While these concepts are generally well described in the domain of music they are less commonly associated with information displays for the other senses.

Temporal auditory metaphors provide some advantages over visual temporal metaphors. Sound has been identified as a useful way for monitoring real time data as audio fades nicely into the background but users are alerted when it changes (Cohen 1994). Kramer makes many other observations about sound (Kramer 1994). Other objects do not occlude sounds. Therefore, an object associated with the sound does not have to be in the field of view for the user to be aware of it. Sounds act as good alarms and can help orientate the user's vision to a region of interest. Auditory signals can often be compressed in time without loose of detail. Because of the high temporal

resolution of the auditory sense, events can still be distinguished.

Many haptic perceptions also require an integration of both spatial and temporal properties and it is expected that many temporal auditory metaphors can be directly transferred to the haptic domain.

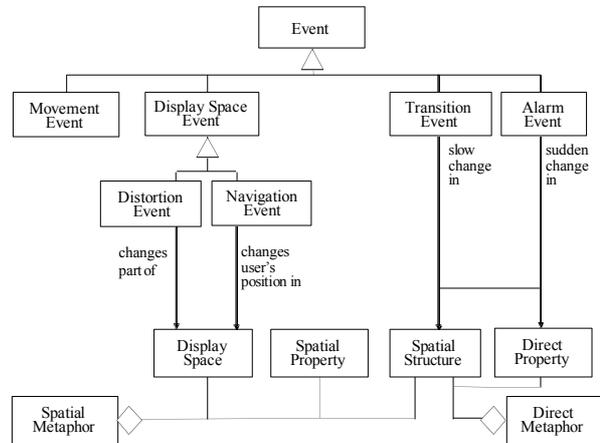


Figure 18. The different types of events used to categorise Temporal Metaphors.

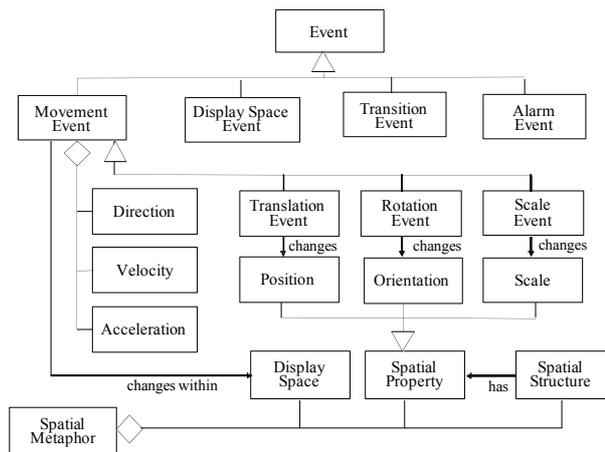


Figure 19. Movement events may have properties of direction, velocity and acceleration. Movement events are defined in terms of the spatial properties of position and orientation

7 Discussion

This paper has described a conceptual model of the multi-sensory, perceptual design space called the MS-Taxonomy. This model has been designed to maximise the reuse of concepts across sensory boundaries. The taxonomy has been derived using a typical software engineering approach that focuses on modelling domain concepts and describing aggregation and inheritance relationships (Blaha et. al. 2005). This technique originally came from the realm of semantic modelling in the field of artificial intelligence, where aggregation relationships are more simply described as "has-a" and inheritance relationships as "is-a" (Sowa 1991).

Multiple levels of abstraction are used and at the higher levels of abstraction the same terminology can be used for describing the haptic, visual and auditory design space. The key difference to this conceptual model of the perceptual design space is that the abstractions are not based on sensory modalities but rather on temporal, spatial and direct metaphors that are common across all senses.

In software engineering terms the MS-Taxonomy allows a designer to consider reuse of designs at both an abstract architectural level and also a more detailed component level. These reusable patterns can be discussed independently of the sensory modality used in the display. This allows for the same design pattern to be implemented and directly compared between senses.

Of course whether the MS-Taxonomy provides designers with a best division of the multi-sensory design space is arguable. However, the structure has proved useful for structuring both a design process and a set of guidelines that assist and guide designers of multi-sensory displays (Nesbitt 2003).

This conceptual model is not intended to invalidate other models of the visual and auditory design space. However, what it does demonstrate is the very different modelling outcomes that can result, even in the same domain, when the aims and context of the modellers are different.

8 References

- Bertin, J. (1981) "Graphics and Graphic Information Processing". Readings in Information Visualization: Using Vision to Think. S. K. Card, J. D. Mackinlay and B. Shneiderman. San Francisco, USA, Morgan Kaufmann, pp. 62-65.
- Blaha, M., Rumbaugh, J., (2005), Object-Oriented Modeling and Design with UML. Pearson Education Inc, New Jersey.
- Blattner, M.M., Sumikawa, D. et al. (1989) "Earcons and Icons: Their Structure and Common Design Principles." Human Computer Interaction 4(1). pp. 11-14.
- Bly, S., (1994), "Multivariate Data Mappings". Auditory Display: Sonification, Audification and Auditory Interfaces. G. Kramer, Addison-Wesley Publishing Company. XVIII: 405-416.
- Brooks, F. P., Ouh-Young, J. M. et al. (1990) "Project GROPE- Haptic Displays for Scientific Visualization." Computer Graphics 24(4): 177-185.
- Card, S.K. and Mackinlay, J.D. (1997) "The Structure of the Information Visualisation Design Space". Proceedings of IEEE Symposium on Information Visualization, Phoenix, Arizona, USA, IEEE Computer Society.
- Card, S. K., J. D. Mackinlay, et al., Eds. (1999) Information Visualization. Readings in Information Visualization. San Francisco, California, Morgan Kaufmann Publishers, Inc.
- Cohen, J. (1994), "Monitoring Background Activities". Auditory Display: Sonification, Audification and Auditory Interfaces. G. Kramer, Addison-Wesley Publishing Company. XVIII: 499-534.
- Friedes, D. (1974) "Human Information Processing and Sensory Modality: Cross-Modal functions, Information Complexity, Memory and Deficit." Psychological Bulletin 81(5): 284-310.
- Gaver, W.W. (1986) "Auditory Icons: using sound in computer Interfaces." Human Computer Interaction 2: 167-177.
- Gaver, W. W. (1993). "What in the world do we hear? An ecological approach to auditory source perception." Ecological Psychology 5(1): 1-29.
- Gaver, W. W. (1994) "Using and Creating Auditory Icons". Auditory Display: Sonification, Audification and Auditory Interfaces. G. Kramer, Addison-Wesley Publishing Company. XVIII: 417-446.
- Goldstein, E. B. (1989). Sensation and Perception, Brooks/Cole Publishing Company.
- Granlund, A., D. Lafreniere, et al. (2001). "A pattern-supported approach to the user interface design process". 9th International Conference on Human Computer Interaction, New Orleans, USA.
- Humphrey, W.S. (2000). "A Discipline for Software Engineering". Boston, Addison Wesley.
- Kramer, G. (1994) An Introduction to Auditory Display. Auditory Display: Sonification, Audification and Auditory Interfaces. G. Kramer, Addison-Wesley Publishing Company.
- Nesbitt, K. V., Gallimore, R. et al. (2001) "Using Force Feedback for Multi-sensory Display". 2nd Australasian User Interface Conference AUIC 2001, Gold Coast, Queensland, Australia, IEEE Computer Society.
- Nesbitt, K. (2003), "Designing Multi-Sensory Displays for Abstract Data". Ph.D. Thesis, Information Technology, Science. Sydney, University of Sydney.
- Sekuler R. and Blake R., (1990), "Perception". McGraw-Hill Publishing Company. New York, USA.
- Soukup, T. (2002) Visual data mining: techniques and tools for data visualization and mining. New York, John Wiley & Sons.
- Sowa, John F., ed. (1991) *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, Morgan Kaufmann Publishers, San Mateo, CA, 1991.

Process Modelling: The Deontic Way

Vineet Padmanabhan¹, Guido Governatori¹, Shazia Sadiq¹, Robert Colomb¹ & Antonino Rotolo²

¹ School of Information Technology & Electrical Engineering
The University of Queensland, Queensland, Australia
Email: [vnair,guido,shazia,colomb]@itee.uq.edu.au

² CIRSIFID, University of Bologna, Bologna, Italy
Email: rotolo@cirsifid.unibo.it

Abstract

Current enterprise systems rely heavily on the modelling and enactment of business processes. One of the key criteria for a business process is to represent not just the behaviours of the participants but also how the contractual relationships among them evolve over the course of an interaction. In this paper we provide a framework in which one can define policies/ business rules using deontic assignments to represent the contractual relationships. To achieve this end we use a combination of deontic/normative concepts like *proclamation*, *directed obligation* and *direct action* to account for a deontic theory of commitment which in turn can be used to model business processes in their organisational settings. In this way we view a business process as a *social interaction process* for the purpose of doing business. Further, we show how to extend the i^* framework, a well known organisational modelling technique, so as to accommodate our notion of deontic dependency.

Keywords: Business, Enterprise and Process modelling.

1 Introduction

One of the issues which was heavily debated during the panel discussions of AAMAS-04 (Singh 2004) was with regard to considering business process modelling/management as a killer application for agents. It was decided that in-order to make progress in this direction it was inevitable to characterise problems within business process modelling/management where agents can apply. In this paper we make such a move whereby we consider an agent to represent a real world business partner with its own local business rules and configurations. The collaboration between the different partners is made possible through *normative co-ordination*, i.e., the idea that agents can achieve flexible co-ordination by conferring normative positions like duties, permissions and powers to other agents. In this way we can view the partners involved in a business scenario as multiple agents who might collaborate by creating *commitments* using normative concepts such as obligation, proclamation and so on but at the same time retain their autonomy. This in a way is similar to how a Multi-Agent System (MAS) is defined using abstractions like team and commitments from the perspective of organisations and societies (Conte & Dellarocas 2001, Pitt 2005).

The definition of a business process we adopt in this paper is more general in the sense that we consider a business process as a special kind of *social interaction process*. A *social interaction process* is a temporally

ordered, coherent set of events and actions, involving one or more communication acts that may create commitments, perceived and performed by agents, and following a set of rules, or protocol, that is governed by norms, and that specifies the type of the interaction process (Wagner 2003).

Based on the above definition of a business process, we develop a logical framework based on multi-modal logic to capture the normative positions among agents in an organisational setting. We do not take into account any temporal considerations in this work as it remains part of future work. Also commitment is not taken as a primitive but is rather defined in terms of directed obligations, i.e., an agent's obligation towards another agent. For example, the obligation to pay for delivered goods is directed, so to say from the buyer to the seller in a business situation like trade where different agents like *buyer*, *seller*, *transport company*, *customs offices* etc. exist. The reasons for not adopting commitments as a primitive is that (1) we want to show that deontic logic can be used to capture an agents *obligation* to another and (2) thereby show that the argument made in (Singh, Chopra, Desai & Mallya 2004) which states that *commitments fare better than traditional deontic logic because deontic logic disregards an agents obligation to another agent* is not true. Two other important concepts we use in our framework are that of *proclamation* and *direct action*. Proclamation is seen as a particular type of speech-act (communication act) that helps the agents to create normative positions involving other agents. It is similar to the role speech acts play in *language-action* approaches to work-flow management as given in (Goldkuhl 1996). Proclamation is used to cover all those speech acts by which an agent makes a statement expressing a certain proposition with the aim of making the proposition true. In other words, different types of speech acts like *directives* (permitting, requesting, forbidding etc.), *Commissives* (agreeing, offering, promising etc.), *Constatives* (announcing, disagreeing, informing etc.) etc. are considered as instances of just one speech act like proclamation. The advantage of this approach is that one need to worry only about the specific semantics corresponding to the *proclamation* operator rather than for each type of speech act. On the other hand direct action is concerned with relationships between agents and states of affairs they want to realise/bring about. We discuss more about these in the coming sections.

Let us put together the different concepts explained above with the help of an example. Consider the work-flow in Figure 1. depicting a simple scenario of managing after sales service.

This process is assumed to execute in a technology which represents a shared space between various business partners and stake-holders of the overall business process. The 4 actors (roles) within the scenario are that of *Customer*, *Retailer*, *Manufacturer* and *Technician*. In such a set up it is natural to think of commitments as being made from one actor to another to achieve some goal and these commitments could be in the form of obligations. For ex-

Copyright ©2006, Australian Computer Society, Inc. This paper appeared at Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Australia. Conferences in Research and Practice in Information Technology, Vol. 53. Markus Stumptner, Sven Hartmann and Yasushi Kiyoki, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

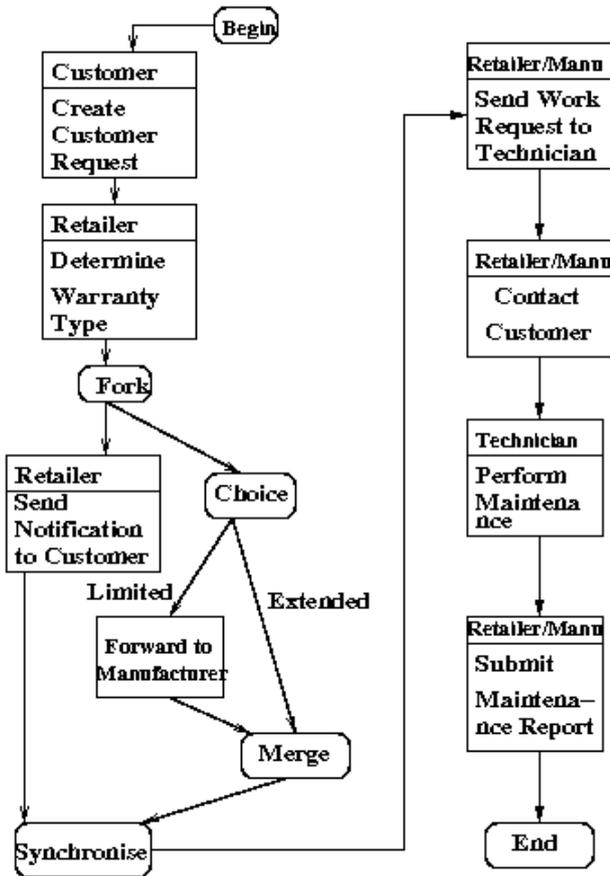


Figure 1: After sales service work-flow

ample, *Send work request to technician* and *Contact Customer* are two activities which may be allocated to either the retailer or the manufacturer depending on the particular instance. Sending request to technician may involve an internal activity of first finding the most appropriate technician. For instance, the retailer/manufacturer is normally *obliged* to issue several RFQs (Request For Quote) to several technicians and choose from among them. The reason is that the retailer/manufacturer wants to get the best quote. The technicians are *obliged* to respond to an RFQ because it represents a potential business that he/she needs. Hence we can say that a particular technician t undertakes an obligation towards the retailer/manufacturer to perform job j while the retailer/manufacturer undertakes towards the technician the obligation of paying the price p . Such obligations are called *directed* obligations and we explain more about their logical properties in section 2. In this way we can define commitments in terms of directed obligations from one party to another. In general, if we have two actors (roles) representing a *debtor* and the other a *creditor* in a particular exchange relationship it is natural to think of a commitment as a directed obligation from the debtor to the creditor regarding a particular condition that in effect the debtor promises to *bring about*. It is also the case that the different actors could have reciprocal obligations whereby they can make effective contractual relationships/joint commitments between them. We show how a combination of notions related to *obligation*, *proclamation* and *action* accounts for a deontic theory of commitment which in turn can be used to model business processes in their organisational settings. Not only does such a model enable to describe and analyse commitments that exist between the actors of a problem domain but also helps to achieve coherent behaviour in interaction processes. We achieve this by defining various operations that can be performed to create and manipulate commitments that would help the different actors involved to create various normative relations needed to co-ordinate

their behaviours for the overall success of a business process. In this way, similar to (Taveter & Wagner 2001a), we view a business process as a social interaction process for the purpose of doing business.

Our *deontic approach* to process modelling/management is based on works like (Barbuceanu, Gray & Mankovski 1999, Taveter & Wagner 2001a, Yu & Mylopoulos 1993, Yu & Mylopoulos 1994) where it is often argued that organisations are made up of social actors who have goals and interests which they pursue through a network of relationships with other actors¹. Hence a *richer model of business process should include not only how work products (entities) progress from process step to process step (activities) but also how the actors performing these steps relate to each other intentionally, i.e., in terms of concepts such as goal, belief, commitment etc..* Such process models conveys a deeper understanding of a business process by focusing on the intentional dependencies among actors which is extremely important for modelling business processes between enterprises that consists of the steps of analysis and design. Following this paradigm, in this work, we develop a formal representation based on a deontic approach to capture the normative dependencies between the different actors in a business process scenario. The formal representation allows us to achieve coherent behaviour in the interaction process where the rules of engagement are dynamically and frequently changing.

The paper is structured as follows. In section 2 we outline the various ingredients of our logical framework needed to represent the different aspects involved in a social interaction process (business process). In section 3 we propose our theory of commitment based on a combination of proclamation, obligation and direct action. The next section (section 4) demonstrates an example scenario in which we show our working model. Section 5 shows some general rules of engagement between different actors in an organisational setting. Section 6 makes a comparison of our model with that of t^* and extends the later to accommodate normative dependency. Section 7 talks about related as well as future work.

2 Institutional Agency

As mentioned in the previous section business processes exist in social organisational settings wherein interaction between agents takes place in a social context. Hence normative concepts are essential for understanding and controlling coherent interaction between agents and other systems. For this paper we take as background the well-known Kanger-Lindahl-Pörn (Kanger 1972, Lindahl 1977, Pörn 1977) logical theory to account for agency and organised interaction (see (Elgesem 1997)). Our starting point is to take advantage of some recent contributions (Santos, Jones & Carmo 1997, Jones & Sergot 1996, Pitt 2005), which have enriched this framework with some substantial refinements. As we have alluded to, the notion of agency is described in a multi-modal logical setting. Despite some well-known limitations (see (Elgesem 1997, Royackers 2000)), such an approach is very general since actions are simply taken to be relationships between agents and states of affairs, and very flexible since it allows for the easy combination of actions and concepts like powers, obligations, beliefs, etc. It also permits to provide a simple conceptual analysis of the structure of organisations of agents. As recently pointed out regarding the design of computerised multi-agent systems, such a multi-modal logic “[is] a means of supplying an intermediate level of description, falling somewhere between [...] ordinary-language account of what a system [...] is supposed to be able to do and [...] the level of implementation” (Pitt 2005).

¹We use the term agent/actor interchangeably.

2.1 The Logical Framework

We first provide the basic ingredients that make up a theory of institutional agency and later show how a notion of commitments can be captured by combining these basic ingredients. We start with the idea of personal and direct action to realise a state of affairs, formalised by the modal operator E : $E_x A$ means that the agent x brings it about that A . For example, suppose that A represents a situation in Figure(1) where *a particular technician, t_1 , can do the maintenance job on a specific date*. Then it could be that the retailer r brings it about that A by sending a request to the technician t_1 . Different axiomatisations have been provided for E but almost all include the following schemas.

$$E_x A \rightarrow A \quad (1)$$

(1) is recognised as valid by almost all theories of agency. It is nothing but the usual axiom T of modal logic, and it expresses the successfulness of actions that is behind the common reading of “bring about” concept. We reformulate this axiom as $E_X A \rightarrow A$ to represent a set of agents X . $E_{\{x\}} = E_x$ when the set of agents is a singleton. $E_x A$ can have the form $E_x \text{sendGoods}(p)$, (where A is an action predicate denoting a specific action), meaning agent x sends the goods p . Here agent x executes by itself the action A . Most of the examples in this paper interprets A as an action predicate. It should be noted that this general approach to the treatment of action does not take into consideration state change and temporal dimension and is focused only on the agent concerned and the states of affairs that result from his/her actions. It is not a drawback as far as this work is concerned because when specifying rules/policies for normative co-ordination in an organisational setting where multiple actors are involved, it may be that only the end result together with which actors brings it about is important.

One other axiom advanced in (Santos et al. 1997) – and adopted here – to characterise specifically the action operator E is

$$E_x E_y A \rightarrow \neg E_x A \quad (2)$$

which corresponds to the idea that the brings-it-about operator expresses actions performed directly and personally. Compared to $E_x A$ which pertains to *individual* agent positions $E_x E_y A$ denotes *interpersonal control* positions and it is this ability to iterate action operators that could be seen as a benefit for having a general theory of action. From a process point of view the above axiom states a principle of rationality for modelling co-ordination between different actors in a process model. For instance, such an axiom can be used to show that an actor x delegates an actor y to bring about a condition. It is counterintuitive that the same agent brings it about that A and brings it about that somebody else achieves A .

2.2 Obligation

Our logical framework incorporates obligations. We use O as a directed deontic operator indexed by a set of agents to represent obligation. We write $O_x E_{\{y\}} A$ to mean that agent y has towards agent x the obligation of realising A . As in the case of E we need to sketch a suitable axiomatisation for O . We cannot use Standard Deontic Logic (SDL) for this purpose as it has been shown in (Royackers 2000) that SDL is not adequate for combining deontic and action operators. The reason is that SDL supports the following implications which is not acceptable from a process modelling view point

$$O_y E_x A \rightarrow O_y A \quad (3)$$

$$O_z E_x E_y A \rightarrow O_z E_y A \quad (4)$$

For instance, let x be a retailer, y a customer and A is the condition *sendGoods*. Then the retailers obligation towards the customer to bring about the condition

sendGoods should not entail that *sendGoods* is in general obligatory. Similarly in the case of the second implication suppose that we have a scenario where in addition to the retailer and customer a manufacturer z is also involved and A is the condition *buyProduct*. Now the retailers obligation towards the manufacturer to bring about the condition that the customer buys a particular product does not entail that the customer has a personal obligation to the manufacturer to buy that product. To avoid the above problems we only consider the following axioms for a logic of obligation

$$(O_x A \wedge O_x B) \rightarrow O_x (A \wedge B) \quad (5)$$

$$O_x A \rightarrow \neg O_x \neg A \quad (6)$$

2.3 Proclamation

The link between speech acts theory and normative positions has been under investigation for some time now (cf. (Jones 1990, Castelfranchi, Dignum, Catholijn & Treur 2000, Singh 1999, Colombetti 2000)). (Gelati, Rotolo, Sartor & Governatori 2004) defines proclamation as a special type of speech act (communication act) dealing with all those acts by which a subject makes a statement expressing a certain proposition and this statement has the function of making this proposition true. In this way it can be seen as a *see-to-it-that* modality indirectly representing a speech act and can be formalised by the modal operator *proc*. As for E , *proc* will be indexed by sets of agents and therefore $proc_X A$ means that the members of X jointly proclaim A . As before when X has only one element x , $proc_X A$ means that A is proclaimed personally by x . Its logic is characterised by some very minimal properties: it is closed under logical equivalence, i.e., $A \equiv B \rightarrow proc A \equiv proc B$ and includes at least the axiom

$$(proc_X A \wedge proc_X B) \equiv proc_X (A \wedge B). \quad (7)$$

Of course, *proc* is not necessarily successful. Whether it is successful or not, within an organisational setting s , depends on whether s makes it effective by means of appropriate *counts-as* rules. We talk more about the counts-as rule in the next section.

Since we define commitments in terms of directed obligation and commitments typically arise from certain communication acts, we can use *proc* to model a communication act through which a particular agent conveys his/her obligation towards another. For example in the scenario depicted in Figure1 $proc_r(O_t E_{\{r\}} A)$, conveys a communication act made by the retailer (r) meaning that the retailer is obligated towards the technician (t) to bring about a certain condition, for instance to pay a specific amount of money. This proclamation by r could be seen as r 's attempt to commit itself towards t . In a similar manner to model a communication act expressing a joint commitment between two parties involved we can use $proc_{\{x,y\}}$. We discuss more about joint commitments in the coming sections. Also, as we have observed, proclamations are not necessarily effective in the sense that when an agent x proclaims that A , x brings it about that A is dependent on the concerned organisation. In the next section we show how we can achieve this result.

2.4 The counts-as Rule

In the previous sections we described the main ingredients needed to develop a theory of institutional agency. The intuition behind such an exercise was to show how agent-oriented approaches to normative agency can be used in the domain of process modelling/management. But we need some more material to complete the picture. We need a way to express that certain facts hold in the context of an institution. For instance, it is normal in a norm-governed institution that designated agents are empowered to create

*institutional facts*² by performing certain types of actions. Hence each organisation needs rules about which agents are empowered to assign rights, or to alter existing rights. Such power assignment rules can often be represented using a *counts-as* structure to denote the context in which they operate. In a business process scenario this is important as the context denotes a group that contains the participating agents usually in different roles thereby enabling the user to specify those rules that *count as* effective in a particular context. We represent this contextual structure using a conditional connective, \Rightarrow_s , to express the *counts as* connection in an institution s . It should be noted that this conditional connective is used just as a symbol of representation and has got nothing to do with the formalism developed in Jones and Sergot (Jones & Sergot 1996). The intuition behind the above definition of the counts-as link is that we want to capture the idea that counts-as rules may specify when an institutional act (e.g., a contract made by person x in the name of person y) has the same effects of another institutional act (e.g., a contract made by y).

Another nice feature of the counts-as rule is in its ability to express various forms of *normative delegation* when combined with *proc*. Since *proc* is not successful, its effectiveness is provided by the institution assuming rules such as

$$proc_x A \Rightarrow_s E_x A. \quad (8)$$

Such a combination can be used to capture two forms of delegation such as

$$proc_y (proc_x A) \Rightarrow_s E_y (proc_x A) \quad (9)$$

$$proc_y (E_x A) \Rightarrow_s E_y (E_x A) \quad (10)$$

where the proclamation or the action of y count as the proclamation or the action of somebody else. (9) conveys the meaning that when y proclaims that x proclaims that A , this counts as y making so that x proclaims that A (here x is the principal and y is the representative). For example, *Determine Warranty type* in Figure(1) is an automated activity which interprets the conditions of the after sales service agreement and appropriately routes the subsequent activities to the correct process role (retailer or manufacturer). There can be a condition in the agreement which states that if the purchase is made in the last 12 months, then the warranty will be covered by the manufacturer. However, the retailer provides extended warranty services, and if the problem is reported in the second or third year of the purchase, then the retailer will provide the maintenance service. Consider a particular situation in our scenario where the retailer r represents the manufacturer m with respect to informing the customer c that *if the purchase is made in the last 12 months then the warranty will be covered by the manufacturer*. This could be seen as a proclamation from the manufacturer denoting a business policy he/she follows to do business. We can formally define this policy as

$$proc_m (O_c E_m (coverWarranty(1^{st} year))) \quad (11)$$

where $(O_c E_m (coverWarranty(1^{st} year)))$ is the content of the manufacturer's proclamation denoting his/her obligation to the customer to cover warranty for the first year. Hence (9) can be reformulated as

$$proc_r (proc_m (O_c E_m (coverWarranty(1^{st} year)))) \Rightarrow_s E_r (proc_m (O_c E_m (coverWarranty(1^{st} year)))) \quad (12)$$

Therefore as far as c is concerned with the reading that r 's proclamation about m 's business policy that c 's warranty will be covered by m for the first 12 months counts as r 's making so that m proclaims the policy.

²For a distinction between *social facts* and *institutional facts* refer (Taveter & Wagner 2001a).

Let us see how such rules of delegation gets used in our scenario. Suppose that (12) is a business rule³ representing certain conditions of the after sales service agreement. Once *Create Customer Request*, Figure(1), is performed by the customer using the designated service available through the portal it is up-to *Determine Warranty Type* to interpret conditions of the after sales service agreement and route subsequent activities accordingly. We will see how the conditions/constraints in rules like (12) gets interpreted/reasoned about. From $E_r (proc_m (O_c E_m (coverWarranty(1^{st} year))))$ (the consequent of (12)) and (1) we get

$$(proc_m (O_c E_m (coverWarranty(1^{st} year)))) \quad (13)$$

Similarly from (13) and (8) we get

$$E_m (O_c E_m (coverWarranty(1^{st} year))) \quad (14)$$

Again by applying (1) to (14) we get

$$O_c E_m (coverWarranty(1^{st} year)) \quad (15)$$

conveying that the manufacturer is obliged to the customer to cover warranty for the first year and accordingly *Determine Warranty Type* will route further activities to the manufacturer if the customer request states that his/her purchase was done in the last 12 months. More complex constraints can be easily added to such rules as we show later in this paper. Also, our choice of the consequent part of (12) was arbitrary as we would get the same result with the antecedent too.

A representation like (10) is necessary when the representative substitutes a principal which would not be able to perform directly the activity delegated to the representative. Also when applied to action descriptions, formulas like

$$E_x A \Rightarrow_s E_x B \quad (16)$$

$$E_x A \Rightarrow_s E_y B \quad (17)$$

represent respectively x 's institutional power to produce B when A is realised and x 's power to perform an action as if something else were made by y (see (Jones & Sergot 1996)).

The last notion we have to deal with is that of *Declarative power*. The concept of declarative power is common in many normative systems and consists in the capacity of the power-holder of creating institutional facts, simply by "proclaiming" them. But as pointed out earlier, proclamations are not necessarily effective and when an institution provides for the effectiveness of a proclamation we say that the subject of the proclamation has a declarative power. The following definition holds

$$DeclPow_{\{x\}} A =_{def} proc_{\{x\}} A \Rightarrow E_{\{x\}} A \quad (18)$$

conveying the meaning that an agent x has the declarative power of producing A means that if x proclaims that A then x produces A . In a similar manner to show that every couple of actors has the power of establishing any obligation between them simply by proclaiming it we have the following representation

$$DeclPow_{\{x,y\}} (O_y E_{\{x\}} A) \quad (19)$$

3 Commitments via Proclamation, Obligation and Direct Action

In the previous sections we outlined various constructs needed for a theory of normative agency. In this section

³Business rules are statements that express (certain parts of) a business policy, defining business terms, and defining or constraining the operations of an enterprise (Taveter & Wagner 2001a).

we show how combining these different concepts we can arrive at a notion of commitment which is needed to effectively form a contract between two participating entities in a business scenario. In (Singh 1999, Yolum & Singh 2004) commitment is treated as first-class abstract objects where a base level commitment is represented as a four-place relation, $C(x, y, G, p)$, denoting a commitment from x toward y to bring about a condition p in the context of G . Here x is the agent who is committed (*debtor*) and y is the agent who receives the commitment (*creditor*). We could express this in our formalisation by a two step process without taking commitment as a primitive as follows,

$$O_y E_{\{x\}} p \quad (20)$$

with the reading that x is obliged to y to bring it about p . In order to show the commitment aspect we combine *proc* with the directed deontic operators above to give us

$$proc_{\{x\}}(O_y E_{\{x\}} p) \quad (21)$$

Here, the proclamation is x 's attempt to commit itself towards y and thereby makes x responsible to y for satisfying p . For a stronger version of commitment we need to provide (21) with some additional support. It has been pointed out in (Gelati et al. 2004) that proclamations are not necessarily effective in the sense that when an agent x proclaims that p , x brings it about that p only if the institution provides for this result. But this is not a problem in our set up as we can give a rule like

$$proc_{\{x\}} p \Rightarrow_{SCM} E_{\{x\}} p \quad (22)$$

which conveys the meaning that in the context of Supply Chain Management (SCM) x 's proclamation of p counts as x bringing it about p . For example, let x denote a *supplier*, y a *customer* and p denote a condition like *sendGoods*(p). Then from (21) and (22) we get the reading that, in the context of an SCM, the supplier's proclamation regarding his/her obligation towards the customer to bring about the act of sending goods counts as the supplier bringing about the act. In other words the supplier's proclamation regarding his/her commitment towards the customer counts as the supplier realising those commitments. Hence the supplier's commitment towards the customer to send goods results in sending the goods. This formalisation also goes well with the commonsensical view that commitments to other agents represent commitments to oneself to bring it about. In a business process scenario like SCM, (21) and (22) could be seen as *policies* that govern the commitment operations among different stakeholders and are part of the contractual relationships existing between them. They are considered as policies as they differ from the local business rules and configurations that make up a particular partner in a business scenario. In a similar manner (23) shows x 's attempt to *command* y and (24) conveys x 's attempt to *free* itself from an obligation towards y .

$$proc_{\{x\}}(O_x E_{\{y\}} p) \quad (23)$$

$$proc_{\{x\}}(\neg O_y E_{\{x\}} p) \quad (24)$$

3.1 Commitments Through Reciprocal Obligations

In the previous section we saw how a proclamation by a single agent, where a combination of directed obligation and action is involved, could account for base-level commitments of the type $C(x, y, G, p)$. But as pointed out in (Gelati et al. 2004) there could be *multi-lateral proclamations* within an institution where a set of agents is involved in a proclamation with reciprocal obligations as the content of the proclamation. We use such proclamations to substitute the conditional commitments as proposed in (Singh 1999). A conditional commitment $CC(x, y, p, q)$

denotes that if the condition p is satisfied, x will be committed to bring about condition q . In other words conditional commitments are useful when an agent wants to commit only if a certain condition holds or only if the other party is also willing to make a commitment. In our case this condition is achieved through a notion of mutual obligation. It is also the case that such proclamations can be used to denote meta-commitments which in turn are rules that govern the commitment operations as was pointed out in (Singh 1999). Reciprocal obligations are used as the content of such proclamations through which we capture the meta-commitment idea. Actually, in a business process scenario like SCM such joint proclamations carry more sense. For instance, in an exchange relationship between a supplier (x) and a customer (y), we can define a commitment between the two parties through a joint proclamation by combining (21) and (23) as follows;

$$proc_{\{x, y\}}(O_y E_{\{x\}}(sendGoods(p)) \wedge O_x E_{\{y\}}(sendMoney(q))) \quad (25)$$

The proclamation made in (25) shows the joint commitment between x and y by taking x 's obligation towards y to send goods p and y 's obligation towards x to send money q . In other words, there should exist reciprocal obligations between x and y to create such mutual commitments. Elsewhere (Gelati et al. 2004) the term *contract* is given for such commitments. It is also the case that a joint proclamation like (25) boils down to two further committing acts of *offer* and *accept* where x 's offer to y is based on reciprocal obligations between x and y and y accepts this. For instance if x 's offer to y is based on the reciprocal obligation between x and y that x sendGoods(p) and y sendMoney(q) and y accepts to it then this counts as making the commitment. Formally,

$$offer_{\{x\}, \{y\}}(sendGoods(p), sendMoney(q)) \wedge accept_{\{x\}, \{y\}}(sendGoods(p), sendMoney(q)) \Rightarrow_{SCM} proc_{\{x, y\}}(O_y E_{\{x\}} sendGoods(p) \wedge O_x E_{\{y\}} sendMoney(q)) \quad (26)$$

In this way we restrict commitments to the creation of reciprocal obligations. Now as noted in (Gelati et al. 2004) it is possible to define *offer* and *accept* which are basically committing acts in terms of non committing acts like *proposal* and *agree*. For instance

$$proposal_{\{x\}, \{y\}}(sendGoods(p), sendMoney(q)) =_{def} proc_x(O_y E_{\{x\}} sendGoods(p) \wedge O_x E_{\{y\}} sendMoney(q)) \quad (27)$$

conveys x 's declaration whereby he/she proposes not only to have an obligation towards y to do *sendGoods*(p) but also to command y to do *sendMoney*(q) (i.e., (21) and (23)). In similar lines we can show that y *agree* with x when x has already made a proclamation in which a specific contractual content is proposed for y to bring about and y makes a proclamation to commit itself towards x to bring about this content.

$$agree_{\{y\}, \{x\}}(sendGoods(p), sendMoney(q)) = proposal_{\{x\}, \{y\}}(sendGoods(p), sendMoney(q)) \wedge proc_{\{y\}}(O_x E_{\{y\}} sendMoney(q)) \quad (28)$$

From the above discussion we can say that an *offer* takes place in a business process scenario like SCM when the

following condition is satisfied;

$$\begin{aligned} offer_{\{x\},\{y\}}(sendGoods(p), sendMoney(q)) = \\ & proposal_{\{x\},\{y\}}(sendGoods(p), sendMoney(q)) \wedge \\ & \{agree_{\{y\},\{x\}}(sendGoods(p), sendMoney(q)) \Rightarrow_{SCM} \\ & \quad proc_{\{x,y\}}(O_y E_{\{x\}} sendGoods(p) \wedge \\ & \quad O_x E_{\{y\}} sendMoney(q))\} \end{aligned} \quad (29)$$

The main idea of (29) is to show how the contractual relationships among the participants (in this case its between the supplier and customer) in a business process scenario like SCM evolve over the course of an interaction. For instance, x 's proposal to y of a specific contractual content and y 's acceptance of it would create the respective directed obligations between them which in turn leads them to form mutual obligations and thereby arrive at a joint commitment. In other words (29) can be seen as a business policy stating that an offer is made between two stakeholders when there is a proposal and agreement between the two wherein there is mutual commitment regarding a specific contractual content. Such business rules help in outlining the guidelines and restrictions with respect to states and processes in an organisation. They are declarative statements describing *what* has to be done rather than *how* to do it. In a similar manner y 's acceptance with regard to a contractual relationship could be given as

$$\begin{aligned} accept_{\{y\},\{x\}}(sendGoods(p), sendMoney(q)) = \\ & offer_{\{x\},\{y\}}(sendGoods(p), sendMoney(q)) \wedge \\ & proc_{\{y\}}(O_x E_{\{y\}} sendMoney(q)) \end{aligned} \quad (30)$$

indicating y 's agreement with the contractual content. In this manner, by applying a small set of operations like proposal, accept etc. on the combination of directed obligation and action we can represent various contractual relationships of interest in a business scenario. Table 3.1 shows that all the operations defined in (Singh 1999) can be captured in our formalism and some of them in a more intuitive manner. Except for *Assign* all the other operations have a direct reading. The idea of assign is to show that the holder x of the recursive declarative power can exercise his/her power in two ways. The first conjunct shows x 's command over y so that y is obliged to realise A . The second conjunct enables x to transfer to another agent z the same declarative power x possesses i.e., *Assign* transfers a commitment to another creditor within the same context, and can be performed by the present creditor because it is authorised.

4 Bringing them all together

Now we will show how the formal representation developed above can be used to capture the normative dependencies involved in a scenario like the after sales process model as shown in Figure 1. Consider the part where the retailer/manufacture sends notification to the technician. As noted earlier this may involve an internal activity like finding the most appropriate technician which in turn depends on certain agreements reached between a particular technician and the retailer/manufacture. Let us capture this scenario using our framework. It should be noted that in this paper we are interested in the *committed dependency* between the actors i.e., in a committed dependency the dependee will try its best to perform the task because of the fact that the depender would be hurt significantly if the dependency fails⁴. Since in our framework we have

⁴In an *open dependency* if the dependency fails the depender would be affected to some extent whereas a *critical dependency* indicates that some goal of the depender could not be achieved if the dependency fails. For an overview of this classification refer to (Yu & Mylopoulos 1993).

a stronger version of committed dependency in the form of reciprocal obligations (joint commitments) we always have $O_y E_{\{x\}}(X) \wedge O_x E_{\{y\}}(Y)$ denoting the content of the commitment as was given in (25). In the case of our after sales service scenario which includes the retailer and a technician we could state this condition precisely as

$$O_r E_{\{t\}} performed(j) \wedge O_t E_{\{r\}} paid(p) \quad (31)$$

with the reading that the technician t undertakes toward the retailer r , the obligation to perform the job j , while the retailer r undertakes towards the technician t , the obligation of paying the price p . Further, a call for proposal (of making a commitment having content X) by retailer r from any technician $t \in T$ can be represented as $proposal_{\{r\},\{t\}}(X)$. In a similar manner $offer_{\{t\},\{r\}}(X)$ conveys technician t 's offer to retailer r with respect to a commitment having content X and $accept_{\{r\},\{t\}}(X)$ means that the retailer r accepts to a commitment having content X with technician t .

Suppose that the retailer issues a proposal the terms of which states the reciprocal obligations of both parties involved. For instance, the retailers proposal could be that the technician has an obligation to repair a *LG washing machine* (j) and the retailer has the obligation to pay 60 Dollars (p) for it. This could be represented as

$$proposal_r^I(E_{\{t\}} performed(j), E_r paid(p)) \quad (32)$$

From (3.1) we can arrive at the conclusion that (32) refers to the retailers proclamation of a specific proposal. Though this inference is not of much use for this work it is useful when we think about message passing as a kind of speech act. The consequence of (32) is that those technicians who are capable of repairing LG washing machine can make offers. Suppose that one of the technicians returns an offer which is in the form of a counter-proposal.

$$offer_{\{t\},\{r\}}(E_{\{t\}} performed(j), paid(p')) \quad (33)$$

where $p' = 50$ Dollars. Assume that this is the best offer the retailer has received and therefore he/she accepts it. Now the acceptance by the retailer implies his/her agreement because by (29) an *offer* happens when a proposal and agreement is already in place. The acceptance by r could be given as

$$accept_{\{r\},\{t\}}(E_{\{t\}} performed(t), E_{\{r\}} paid(p')) \quad (34)$$

Using (33) and (34) along with (3.1) gives us

$$(proc_{\{r,t\}}(O_r E_{\{t\}} performed(t) \wedge O_t E_{\{r\}} paid(p'))) \quad (35)$$

which shows that the parties have made a joint commitment within the context of *After Sales Service Processing* (ASSP) scenario. Also because of (7), (35) implies

$$proc_{\{r,t\}}(O_r E_{\{t\}} performed(t)) \wedge proc_{\{r,t\}}(O_t E_{\{r\}} paid(p')) \quad (36)$$

By applying the rules provided earlier which shows the effectiveness of a proclamation we can derive from (36) the conclusion that t is obliged to do the job and r is obliged to pay for it. Formally this is given as follows

$$(O_r E_{\{t\}} performed(t) \wedge O_t E_{\{r\}} paid(p')) \quad (37)$$

The above example tells us that agent's behaviour within organisations are governed by social rules that impose obligations over the agents actions. Therefore coordination in organisations and societies cannot be accounted for without considering the social laws of the organisations and the way they constrain behaviours of individual agents. Hence as a first step to make this view practically usable in applications it is important to represent and reason about the obliged behaviours within agents as we have shown above. This in turn would help to explain co-ordination among agents as negotiation about obliged behaviours.

Operation	Meaning	Representation
Create	Instantiates a commitment	$proc_{\{x\}}(O_y E_{\{x\}} A)$
Cancel	Revokes the commitment	$proc_{\{x\}}(\neg O_y E_{\{x\}} A)$
Release	Eliminates the Commitment	$proc_{\{x\}}(\neg O_x E_{\{y\}} A) \wedge proc_{\{x\}}(\neg O_x \neg E_{\{y\}} A)$
Delegate	Shifts the role of x (debtor) to another agent within the same context	$proc_{\{x\}}(proc_y A) \Rightarrow_s E_x(proc_y A)$ $proc_x(E_y A) \Rightarrow_s E_x(E_y A)$
Assign	Transfers a commitment to another creditor within the same context	$RecDeclPow_{\{x\}}(O_x E_{\{y\}} A) =$ $DeclPow_{\{x\}}(O_x E_{\{y\}} A) \wedge$ $DeclPow_{\{x\}}(RecDeclPow_{\{z\}}(O_x E_{\{y\}} A))$

Table 1: Operations on commitments

5 Rules for Deontic Dependency

In this section we provide some general rules that takes care of the deontic constraints to be satisfied between actors. These rules in turn regulates the agent behaviour in an organisational set up. A rule stating the deontic constraints that need to be satisfied to make a proposal between a debtor (x) and creditor (y) can be stated as follows;

$$\begin{aligned}
 & \text{IF } isDebtor(x) \\
 & \quad \text{AND } isCreditor(y) \\
 & \quad \quad \text{AND } (O_y E_{\{x\}} sendGoods(p)) \\
 & \quad \quad \text{AND } (O_x E_{\{y\}} sendMoney(q)) \\
 & \text{THEN } proposal_{\{x\},\{y\}}
 \end{aligned}$$

In a scenario like in (1), we can use the above rule to represent the constraints related to a proposal of a contract issued by the Retailer. For instance, the retailers proposal could be that the technician has an obligation to *repairLG washingmachine*(q) and the retailer has the obligation to pay 60 Dollars(p) for it which could be given in the following way;

$$\begin{aligned}
 & \text{IF } isRetailer(r) \\
 & \quad \text{AND } isTechnician(t) \\
 & \quad \quad \text{AND } (O_r E_{\{t\}} performed(q)) \\
 & \quad \quad \text{AND } (O_t E_{\{r\}} paid(p)) \\
 & \text{THEN } proposal_{\{r\},\{t\}}
 \end{aligned}$$

Similarly, we can give a rule stating the constraints to be satisfied so that an *agreement* could be reached between retailer r and technician t regarding a specific contractual content as follows;

$$\begin{aligned}
 & \text{IF } proposal_{\{r\},\{t\}}(performed(q), paid(p)) \\
 & \quad \text{AND } proc_{\{t\}}(O_r E_{\{t\}} performed(q)) \\
 & \text{THEN } agree_{\{t\},\{r\}}(performed(q), paid(p))
 \end{aligned}$$

i.e., in order for the retailer and the technician to come up with an agreement, initially, there should be a proposal from the retailer to the technician regarding a specific contractual content and the technician makes a proclamation through which he obliges himself to bring about the specific content. It is also possible that the technician t can come up with a better *offer* for the retailer by quoting a different price p' for the job to be performed. A rule for an offer could be given as follows;

$$\begin{aligned}
 & \text{IF } proposal_{\{t\},\{r\}}(O_r E_{\{t\}} performed(q)) \\
 & \quad \text{AND } (O_t E_{\{r\}} paid(p')) \\
 & \quad \text{AND } agree_{\{r\},\{t\}}(performed(q), paid(p')) \\
 & \text{THEN } offer_{\{t\},\{r\}}
 \end{aligned}$$

6 Accommodating Deontic Dependencies in i^*

In the previous sections we developed a framework that provides a normative description of a (business) process

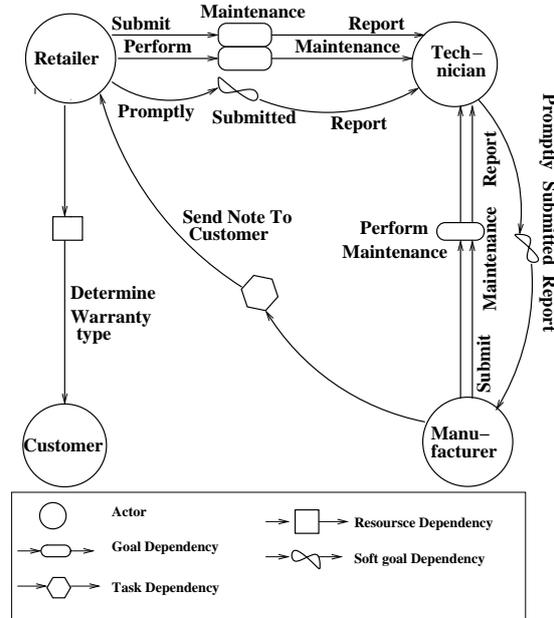


Figure 2: SD-model for the After Sales Service Scenario in Figure 1:

in their organisational settings. In this section we show a dependency diagram similar to i^* in which we can accommodate normative dependency. The i^* Framework (Yu & Mylopoulos 1994, Yu & Mylopoulos 1993), (pronounced *i-star* and stands for *distributed intentionality*), is an organisational modelling technique used by many groups around the world in their research on early requirements engineering, business process design, software development methodologies and many more. The i^* approach enables to describe, model and reason about the goals of systems (business and socio-technical) that involve many different actors and for choosing system architectures that best meet these goals. By explicitly modelling and analysing strategic relationships among multiple actors the approach incorporates rudimentary social analysis into a system analysis and design framework. The framework is based on the **SD** (*Strategic Dependency*) model and the **SR** (*Strategic Rationale*) model wherein the actors are related to each other *intentionally*. Actors depend on each other for goals to be achieved, tasks to be performed and resources to be furnished. Whereas SD-model is used to represent a particular design for a business process the SR-model describes the reasoning that actors have about the different possible ways of organising work, i.e., different configurations of SD networks. Since our main aim is to account for a notion of deontic dependency in the i^* framework we restrict ourselves to the SD-model.

Figure (2) shows a Strategic Dependency model for the after sales service work-flow in Figure (1). As can be seen from the figure the SD model consists of a set of nodes and links. Each node represents an actor, and each link

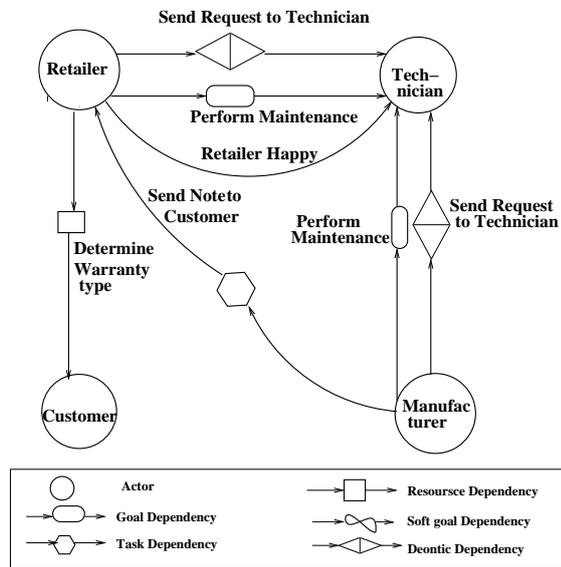


Figure 3: SD-model with Deontic Dependencies for the After Sales Service Scenario in Figure 1:

between two actors indicates that one actor depends on the other for something in order that the former may attain some goal. Four types of dependencies are distinguished in an SD-model between a **dependor** (the depending actor) and a **dependee** (actor who is depended upon). The object around which the dependency relationship centres is called the **dependum**. In a *goal-dependency* a dependor depends on the dependee to bring about a certain state in the world wherein the dependee is free to choose *how* to accomplish the goal. For instance, in Figure (1), the *Retailer/manufacturer* depends on the *Technician* in-order to perform maintenance. The retailer/manufacturer is only concerned about the outcome of *perform maintenance* and doesn't care how the technician achieves the goal. Another goal dependency in Figure (1) is that of *submit maintenance report* between the retailer/manufacturer and the technician. Similarly, in a *task-dependency*, though a dependor depends on the dependee to carry out an activity (the dependum), the activity specification constrains the choices of the dependee on *how* the task is to be performed. As an example of task dependency, the *Manufacturer* depends on the *Retailer* to send notification to the customer regarding the warranty. This is so because the *Manufacturer* wants to have the warranty determined (limited/extended) according to the well defined instructions outlined by it. In a *resource dependency*, the dependor depends on the dependee for the availability of an entity (physical or informational). The dependency between the *Retailer* and the *Customer* to determine warranty type is a resource dependency where the dependum is a piece of information related to the date on which the customer purchased the particular product. In a *softgoal dependency* a dependor depends on the dependee to bring about a condition in the world wherein the criteria is not precisely defined as in the case of hardgoal dependency. The dependee has a number of ways for achieving the goal and the dependor indicates which combination of choices would sufficiently meet the desired subgoal. Its usually considered that a softgoal is *satisfied* rather than satisfied. In Figure (1) *promptly submitted report* is a softgoal dependency between the retailer/manufacturer and the technician. We will add one more dependency, deontic dependency, to this list as shown in Figure (3) so that the logical framework provided in the previous sections can be used to represent and reason about such dependencies. It should be noted that we avoided some relations in Figure (3) which is already explained in Figure (1) so as not to have the figures cluttered. *Send Request*

to *Technician* is a deontic dependency between the retailer/manufacturer and technician because as mentioned earlier the retailer/manufacturer has the right to choose the most appropriate technician based on certain agreements they have reached as a result of mutual obligation. The dependum, i.e., the activity of sending work request cannot be reduced to any other dependency as it involves obliged behaviours.

Now we will show how to capture the intentional dependencies of the SD-model using our framework developed in the previous sections. In the SD-model the external actor relationships as outlined above are characterised in terms of more basic intentional concepts like belief, goal, ability and commitment. We represent the intentional dependencies as meta-commitments by using the notion of reciprocal obligation as follows; Let us consider a commitment, $c = C(x, y, G, p)$, as given in (Singh 1999) wherein c is base-level if p does not refer to any other commitment and c is a meta-commitment if p refers to a base-level commitment. In the case of i^* framework, for a goal dependency, the condition p is an assertion $achieve(g)$ representing the goal g to be achieved by an agent. For a task dependency on task t , p is $done(t)$ and for a resource dependency on resource r , p is $avail(r)$. In our model we can represent these dependencies in terms of reciprocal obligation of the dependor towards the dependee to satisfy the condition p . For instance, in our model, in the case of multi-lateral proclamations (for example (25), the condition p (which is the content of the proclamation) is always defined in terms of directed obligations i.e. the satisfaction of condition p is based on reciprocal obligation between the dependor and the dependee. This means the commitments arrived at by such proclamations are not at base-level but are meta-commitments since p involves the obligations of both parties involved which in turn leads to a joint-commitment. Hence in a way we can say that in our model we represent an intentional dependency as a meta-commitment/reciprocal obligation of the dependor towards the dependee to create commitments to satisfy condition p . To give an example, suppose that *Perform Maintenance* in Figure(2) is a goal-dependency of the retailer on the technician. Then this dependency can be represented as a reciprocal obligation of the technician towards the retailer through which the technician has an obligation towards the retailer to create a directed obligation (upon receiving a request for repair) to achieve the goal *Perform Maintenance*. In a similar manner we can capture the other dependencies in terms of reciprocal obligations.

7 Conclusions and Future Work

We showed how a combination of notions related to *obligation*, *proclamation* and *direct action* accounts for a deontic theory of commitment which in turn can be used to model business processes in their organisational settings. We first outlined a logical framework, which is based on a multi-modal logic, to represent the various deontic concepts. A peculiar feature of our logical framework is in the use of *proclamation* as a unique speech act that can model all other speech acts that characterise an organisation. For instance, in most other approaches, what we modelled as proclamation is represented through different types of speech acts (commissives, permissives, agreements, etc.) where each one is characterised by its own specific semantics. We considered these differences as instances of just one speech act since as far as we are concerned the differences only pertain to the content which is proclaimed. Then we went on to demonstrate how a combination of obligation, proclamation and direct action can account for at-least two types of normative delegation. The most crucial part of this work was developed next, where it was shown how to achieve normative co-

ordination by imposing social constraints/rules in the form of mutual obligations among the agents/actors. One consequence of this approach is that it allows agents to talk to each other based on their sets of obliged behaviours and thereby have a clean approach to negotiation. Another consequence is that by stating them as deontic constraints the co-ordination among agents can be seen as an *exchange of deontic constraints*. Further, we formalised the various operations that can be performed on commitments based on the new framework. Finally, we compared our model with the *i** framework to show that the various dependency relationships can be explained in terms of our work.

An organisational model like ours can be used to capture, support and enforce social patterns of behaviour of business processes operating in open environments. Open societies need mechanisms to systematise, defend and recommend right and wrong behaviour which in turn can inspire trust into the agents/process that will join them. In our model we make use of obligation, commitment etc. as norms to describe such expected behaviour. Also our model is rich enough to cover wide range of contexts for agent interaction. From a workflow point of view, recent works (Russel, van der Aalst, ter Hofstede & Edmond 2005) show that there has been a shift of perspective from *Workflow Control Patterns* and *Workflow Data Patterns* to that of *Workflow Resource Patterns* where modelling of resources (human/non human) and their interaction is of prime importance. We believe that our framework can contribute much to support modelling in the organisational context in which a process operates.

Though we do not address any computational issues in this paper, work is in progress to develop a computational framework based on the logical intuitions we have described here. A computation model based on Defeasible logic has already been proposed in this regard. Defeasible logic has been developed by Nute (Nute 1987) with a particular concern about computational efficiency and developed over the years by (Maher & Governatori 1999, Antoniou, Billington, Governatori & Maher 2000). The reason being ease of implementation (Maher, Rock, Antoniou, Billington & Miller 2001), flexibility (Antoniou et al. 2000) (it has a constructively defined and easy to use proof theory which allows us to capture a number of different intuitions of non-monotonicity) and it is efficient: it is possible to compute the complete set of consequences of a given theory in linear time (Maher 2001). Having such an inference mechanism allows an agent to deduce the logical consequences of given obligations as well as helps in resolving conflicts among obligations (two important directions in which our work could be extended). At the moment, we have provided two extensions of standard Defeasible Logic. The first incorporates the notions of "counts as" and agency, as described in this paper (Governatori & Rotolo 2003, Governatori, Rotolo & Sadiq 2004). The second combines agency, BDI concepts and obligations (Governatori & Rotolo 2004). Our future work will be devoted to developing a unique framework which is able to deal with the cognitive component (BDI concepts), agency, and normative notions ("counts as" and deontic operators).

Before closing down we want to mention some related works that is of importance to this document. (Gelati et al. 2004) is the starting point for this work. But the major difference is in our use of reciprocal obligations to capture the commitment aspect involved in agents. Also, we address our work from a process modelling point of view whereas (Gelati et al. 2004) is concerned with legal reasoning. The same reason applies to (Tan & Thoen 1998). Two other closely related works are (Taveter & Wagner 2001a, Taveter & Wagner 2001b). In those works to the concept of mutual obligation is absent as well as they do not use deontic logic to represent the different normative concepts. (Yolum & Singh 2004, Singh 1999) provided some insights to our approach. They represent commit-

ments as first class abstract objects using a four place relation whereas we show how commitments arise as a result of the obliged behaviours between the different agents. (Yu & Mylopoulos 1994) and (Yu & Mylopoulos 1993) had a major influence on our work and as was shown in the discussion section our framework can accommodate the various dependency relationships outlined there. The advantages of having a fifth dependency in the form *deontic dependency* is worth investigating. (Barbuceanu et al. 1999) uses obligation, permissions and interdictions (OPIs) to reason about the behaviour of social agents. OPIs are modelled by reducing deontic logic to a particular type of dynamic logic and then constraint satisfaction techniques are used to infer consequences and solve conflicts among obligations and interdictions. We believe that this is a good approach for future work though it will not be possible to account for many normative relations that are defined in a deontic logical setting by reducing it to some type of dynamic logic. (Dignum, Vázquez-Salceda & Dignum 2004) is another work similar to ours written from a software engineering perspective.

Acknowledgements

This work was supported by the Australia Research Council under Discovery Project No. DP0558854 on "A Formal Approach to Resource Allocation in Web Service Oriented Composition in Open Marketplaces".

References

- Antoniou, G., Billington, D., Governatori, G. & Maher, M. J. (2000), A flexible framework for defeasible logics, in 'Proc. American National Conference on Artificial Intelligence (AAAI-2000)', AAAI/MIT Press, Menlo Park, CA, pp. 401–405.
- Barbuceanu, M., Gray, T. & Mankovski, S. (1999), 'Role of obligations in multiagent coordination', *Applied Artificial Intelligence* **13**(1-2), 11–38.
- Castelfranchi, C., Dignum, F., Catholijn, M. & Treur, J. (2000), Deliberative normative agents: Principles and architecture, in 'Proceedings of ATAL 1999', Springer, Berlin, pp. 364–378.
- Colombetti, M. (2000), A commitment-based approach to agent speech acts and conversations, in M. Greaves, F. Dignum, J. Bradshaw & B. Chaibdraa, eds, 'Proceedings of the Fourth International Conference on Autonomous Agents, Workshop on Agent Languages and Conversation Policies', Barcelona, pp. 21–29.
- Conte, R. & Dellarocas, C. (2001), *Social Order in Multiagent Systems*, Kluwer Academic Publishers, Boston.
- Dignum, V., Vázquez-Salceda, J. & Dignum, F. (2004), Omni: Introducing social structure, norms and ontologies into agent organisations., in 'PROMAS', Vol. 3346, Springer, pp. 181–198.
- Elgesem, D. (1997), 'The modal logic of agency', *Nordic Journal of Philosophical Logic* **2**, 1–48.
- Gelati, J., Rotolo, A., Sartor, G. & Governatori, G. (2004), 'Normative autonomy and normative co-ordination: Declarative power, representation and mandate', *Artificial Intelligence and Law* **12**, 53–81.
- Goldkuhl, G. (1996), Generic business frameworks and action modelling, in 'Proceedings of the First International Workshop on Communication Modelling', Electronic workshops In Computing.

- Governatori, G. & Rotolo, A. (2003), A defeasible logic of institutional agency, in 'Proceedings of the Fifth International Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC'03)', pp. 97–104.
- Governatori, G. & Rotolo, A. (2004), Defeasible logic: Agency, intention and obligation, in '7th International Workshop on Deontic Logic in Computer Science, DEON 2004', Springer, pp. 114–128.
- Governatori, G., Rotolo, A. & Sadiq, S. (2004), A model of dynamic resource allocation in workflow systems, in 'Fifteenth Australasian Database Conference (ADC2004)', Australian Computer Science Association, pp. 197–206.
- Jones, A. (1990), Towards a formal theory of communication and speech acts, in P. Cohen & M. Pollack, eds, 'Intentions in Communication', MIT Press, Cambridge, Mass.
- Jones, A. & Sergot, M. (1996), 'A formal characterisation of institutionalised power', *Journal of IGPL* **3**, 427–443.
- Kanger, S. (1972), 'Law and logic', *Theoria* **38**, 105–32.
- Lindahl, L. (1977), *Position of change: A Study in law and logic*, Reidel, Dordrecht.
- Maher, M. (2001), 'Propositional defeasible logic has linear complexity', *Theory and Practice of Logic Programming* **1**(6), 601–711.
- Maher, M. J. & Governatori, G. (1999), A semantic decomposition of defeasible logic, in 'Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)', AAAI Press, pp. 299–305.
- Maher, M. J., Rock, A., Antoniou, G., Billington, D. & Miller, T. (2001), 'Efficient defeasible reasoning systems', *International Journal of Artificial Intelligence Tools* **10**(4), 483–501.
- Nute, D. (1987), Defeasible logic, in 'Handbook of Logic in Artificial Intelligence and Logic Programming', Vol. 3, Oxford University Press, pp. 353–395.
- Pitt, J. (2005), *Normative Specifications in Multi-Agent Systems*, John Wiley & Sons.
- Pörn, I. (1977), *Action Theory and Social Science: Some Formal Models*, Reidel, Dordrecht.
- Royakkers, L. (2000), Combining deontic and action logics for collective agency, in J. Breuker et al., ed., 'Legal Knowledge and Information Systems (Jurix)', IOS Press, Amsterdam.
- Russel, N., van der Aalst, W. M. P., ter Hofstede, A. H. M. & Edmond, D. (2005), Workflow resource patterns: Identification, representation and tool support., in 'CAiSE', Vol. 3520, Springer, pp. 216–232.
- Santos, F., Jones, A. & Carmo, J. (1997), Action concepts for describing organised interaction, in 'Thirtieth Annual Hawaii International Conference on System Sciences', IEEE Computer Society Press, Los Alamitos.
- Singh, M. P. (1999), 'An ontology for commitments in multi-agent systems: Towards a unification of normative concepts', *Artificial Intelligence and Law* **7**, 97–113.
- Singh, M. P. (2004), Business process management: A killer app for agents, in 'Proc. Autonomous Agents and Multi-Agent Systems (AAMAS-2004)', IEEE.
- Singh, M. P., Chopra, A. K., Desai, N. & Mallya, A. U. (2004), Protocols for processes: Programming in the large for open systems (extended abstract), in 'OOPSLA Companion', Vol. 39, pp. 120–123.
- Tan, Y.-H. & Thoen, W. (1998), Modelling directed obligations and permissions in trade contracts, in 'HICSS (5)', pp. 166–175.
- Taveter, K. & Wagner, G. (2001a), Agent-oriented enterprise modelling based on business rules., in 'ER', Vol. 2224 of *Lecture Notes in computer Science*, Springer, pp. 527–540.
- Taveter, K. & Wagner, G. (2001b), A multi-perspective methodology for modelling inter-enterprise business processes., in 'ER (Workshops)', Vol. 2465 of *Lecture Notes in computer Science*, Springer, pp. 403–416.
- Wagner, G. (2003), 'The agent-object-relationship meta-model: towards a unified view of state and behavior', *Inf. Syst.* **28**(5), 475–504.
- Yolum, P. & Singh, M. P. (2004), 'Reasoning about commitments in the event calculus: An approach for specifying and executing protocols', *Annals of Mathematics and Artificial Intelligence. Special issue on computational Logic in Multi-Agent Systems.* **42**(4), 227–253.
- Yu, E. S. K. & Mylopoulos, J. (1993), An actor dependency model of organizational work: with application to business process reengineering., in 'COOCS', ACM, pp. 258–268.
- Yu, E. S. K. & Mylopoulos, J. (1994), From E-R to A-R - modelling strategic actor relationships for business process reengineering., in 'ER', Vol. 881 of *Lecture Notes in Computer Science*, Springer, pp. 548–565.

Defining and Implementing Domains with Multiple Types using Mesodata Modelling Techniques

Sally Rice^{1,2}, John F. Roddick¹ and Denise de Vries¹

¹School of Informatics and Engineering
Flinders University of South Australia
PO Box 2100, Adelaide, South Australia 5001
{sallyr,roddick,Denise.deVries}@infoeng.flinders.edu.au

² School of Computer and Information Science
University of South Australia
Mawson Lakes, South Australia 5095
sally.rice@unisa.edu.au

Abstract

The integration of data from different sources often leads to the adoption of schemata that entail a loss of information in respect of one or more of the data sets being combined. The coercion of data to conform to the type of the unified attribute is one of the major reasons for this information loss. We argue that for maximal information retention it would be useful to be able to define attributes over domains capable of accommodating *multiple types*, that is, domains that potentially allow an attribute to take its values from more than one base type.

Mesodata is a concept that provides an intermediate conceptual layer between the definition of a relational structure and that of attribute definition to aid the specification of complex domain structures within the database. Mesodata modelling techniques involve the use of data types and operations for common data structures defined in the mesodata layer to facilitate accurate modelling of complex data domains, so that any commonality between similar domains used for different purposes can be exploited.

This paper shows how the mesodata concept can be extended to facilitate the creation of domains defined over multiple base types, and also allow the same set of base values to be used for domains with different semantics. Using an example domain containing values representing three different types of incomplete knowledge about the data item (coarse granularity, vague terms, or intervals) we show how operations and data structures for types already existing within the mesodata can simplify the task of developing a new *intelligent domain*.

Keywords: Mesodata, intelligent domains, multiply-typed domains, incomplete information, vagueness, coarse granularity, intervals, relational model, hierarchical domains, data integration.

1 Introduction

Integrating multiple sources of data is of vital importance to many enterprises. Within a single organization such integration can lead to better strategic planning and decision making. Integration across multiple organizations leads to more efficiency and quality of

service due to better utilization of information and elimination of redundancy (Zeng 1999).

Integration of sources with different conceptual schemas is not a trivial process. The need to combine attributes defined on different types often leads to loss of information. For example, consider something as simple as a numerical attribute recorded in one source using integers and in another as fixed point values with two decimal places. These representations could be combined by converting the latter into integers, which clearly involves a loss of accuracy in the decimal data. But perhaps the most likely solution when combining these values is to convert the integers to numbers with two decimal places. This may not appear to involve any information loss, but reporting a value like 24 as 24.00 gives it the appearance of an accuracy that it does not have: in fact, we have lost information about the accuracy of the value. A much better solution is to allow the attribute to take values from both types. This concept we call *multiply-typed domains*.

A problem with the former SQL standard (commonly known as SQL-2) is its lack of built-in support for the creation of user-defined complex data types. By this we do not mean merely the ability to create simple domains using the `CREATE DOMAIN` command as it is defined in the SQL standard (see e.g. (Melton & Simon 2002)), but the ability to create domains with complex structure and semantics. This has led to commercial relational database management systems (RDBMS) adding this facility in an ad hoc way as a response to user demands, as, for example, Oracle has done with its extensive `CREATE TYPE` facility (Lorentz & Gregoire 2003). These extensions can be an awkward fit with the concepts of the relational model. Mesodata provides a method for users to implement domains of arbitrary complexity that fits well with the relational model, and there is no reason why the new features offered in SQL:1999, such as the ability to create complex user-defined types, can't be used in the implementation of these domains.

The concept of mesodata – a middle layer of domain definition sitting between the metadata and the data – allows the separation of the definition of the structure from the semantics of a domain. Commonly used data structures (such as lists, graphs and trees) and operations for the manipulation of these data structures are defined in the mesodata layer. These *mesodata types* can then be used to build specific attribute domains: a mesodata type is used to define the structural aspects of the domain, and the domain is then built by populating it with values, and by implementing any additional semantics through additional operations which can utilise the structural operations

Copyright ©2006, Australian Computer Society, Inc. This paper appeared at Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Tasmania, Australia. Conferences in Research and Practice in Information Technology, Vol. 53. Sven Hartmann, Markus Stumptner and Yasushi Kiyoki, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

of the mesodata type.

In the past, the incorporation of additional semantics in attribute domains has generally been accomplished by extending data models such as the relational model for particular types of data. This type of extension includes developments such as temporal databases (Snodgrass 1995), spatial databases (Schneider 1997, Egenhofer & Franzosa 1991) and probabilistic databases (Dey & Sarkar 1996). The advantages that the mesodata approach has over the development of a special-purpose extended data model for each case include making it easier to combine separate extensions into a single data model (as has been necessary for the development of spatio-temporal databases, for example (Abraham & Roddick 1999)), and the ability for reuse of the same data structures with different semantics. We believe that the potential for reuse facilitates the creation of intelligent domains.

Earlier papers on the mesodata concept have argued for the incorporation of mesodata in Database Management Systems, and have shown how domain evolution can be facilitated using mesodata (de Vries, Rice & Roddick 2004, de Vries & Roddick 2004). This paper further defines the mesodata concept by discussing how attributes can be defined over *multiply-typed domains* – domains capable of accommodating multiple types – and showing how the same set of base values can be used for domains with different semantics. We illustrate multiply-typed domains using an example with three types, two whose base values form hierarchies semantically, and one whose base values are numeric intervals. The example we present is of an attribute defined over a multiply-typed domain within a single relation. However, the methods discussed are relevant to the definition of a common schema to be used across different data sources.

The rest of this paper is organised in the following way. Section 2 presents a conceptual model for mesodata and some examples of basic mesodata types, Section 3 introduces multiply-typed domains through an example involving incomplete data, Section 4 discusses how we used these mesodata types to implement this domain as an exemplar of the use of mesodata techniques, and Section 5 provides a conclusion to this paper.

We have tried to be consistent in our use of the terms *domain* and *type* in this paper. In our usage we intend *type* to refer to the *format* of the data, and *domain* to the broader concept of *allowable values*. However sometimes we are constrained to use one or the other due to things outside of our control, such as SQL syntax.

2 Mesodata

Mesodata is a concept that facilitates the implementation of structurally and semantically-rich domains (*intelligent domains*). Key features include a special *mesodata layer* within which structural aspects are defined for common structures such as graphs and trees, and the ability to accommodate domain variability by mapping between different representations (for example, between names and three-byte RGB values for colours). An intelligent domain is built by matching a mesodata type with a *base type* and a *source relation* to hold the specific structural information for the domain. The base type could form a simple domain (such as INTEGER or CHAR(12)), or it could in turn be a domain based on a mesodata type, so we can define graphs of trees, for example. The important difference between mesodata techniques and object-oriented concepts, is that the former introduces the idea of storing complex *domain*

values in the database. Object-oriented databases are concerned with complex *attribute values*.

This section describes a data model for mesodata. We first give a conceptual model and show how it can be incorporated into a relational database, then we present two mesodata types used in the development of our example intelligent domain.

2.1 Conceptual model

Figure 1 shows an entity-relationship model for mesodata, using UML notation as in e.g. (Connolly & Begg 2005).

Domain represents a multiply-typed domain for an *Attribute*. It is composed of one or more *Types*. *Type* is completely specialised into either a *Simple* type (such as INTEGER or CHAR(12)) or a *Complex* type (one built using mesodata types).

A *Complex* type is described by a *Mesodata* type, which has a structure *SRstructure* and a set of *Operations*. Structural details of the *Complex* type are stored in its source relation *SourceRel*, and its base values have a *Domain*. For example, for a graph whose nodes were strings of length 12, the graph's structure would be described in its source relation, and the base domain of its nodes would be CHAR(12). Note that the base *Domain* can itself be *Complex* (to accommodate graphs of lists, for example), and a *SourceRel* may be used for more than one *Complex* type.

A *Mapping* shows how to convert a value from one *Type* of a multiply-typed domain to another. Each *Mapping* has a type *MapType* which can use a function or a lookup table (or a combination of the two) to convert the values.

The implementation of this conceptual model can be separated into four parts:

1. The entities *Mesodata*, *SRstructure* and *Operations* describing the mesodata types form the layer between the metadata and the data described earlier.
2. The entities *Domain*, *Type*, *Mapping* and *MapType* are implemented as tables belonging to a super-user (such as SYS in Oracle), which are protected from direct manipulation by the user.
3. User tables are created as normal, except that attributes with multiply-typed domains must specify their data type as well as their value.
4. There are some hidden tables automatically created when the complex types are defined, i.e. the source relations and look up tables for mappings. These tables should also not be directly manipulable by the user.

2.2 SQL extensions for multiply-typed domains

For illustrative purposes, we offer the following extensions to the syntax of SQL data definition commands. This consists of extensions to CREATE DOMAIN and CREATE TABLE, and a new command CREATE MAPPING.

To allow the same mapping to be used to map more than one type without redefinition, the definition of a mapping is divided into two parts:

- associating a mapping name with a look-up table and/or a mapping function, and
- associating a mapping name with a FROM type and a TO type.

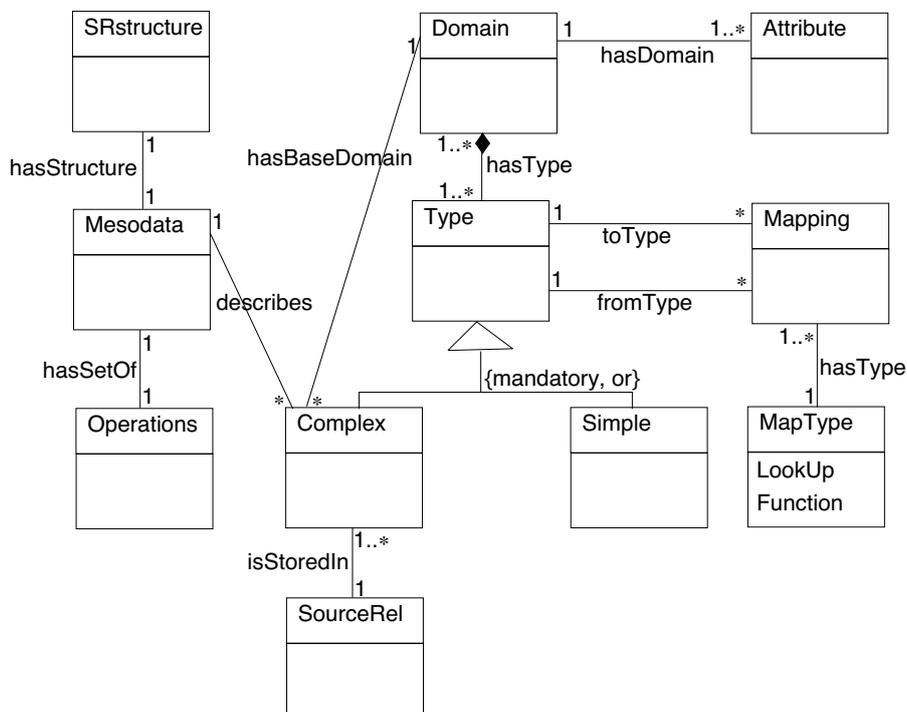


Figure 1: Conceptual model for mesodata

The `CREATE MAPPING` command associates a mapping name with a look-up table and/or a mapping function.

```
CREATE MAPPING mapping_name
  [LOOKUP mapping_rel]
  [FUNCTION function_name]
```

A `MAPPING` clause has been added to the extended `CREATE DOMAIN` command originally defined in (de Vries, Rice & Roddick 2004) to accommodate mesodata types. The `MAPPING` clause defines mappings from the base values associated with a particular `CREATE DOMAIN` command to another type. Because a type may be mapped in more than one way, there may be more than one `MAPPING` clause. The `to_type` can be either a simple or a complex type.

```
CREATE DOMAIN dom
  AS mesodatatype
  OF basedom
  (RETURNS returndom)
  OVER sourcerel {(attribute {,attribute})}
  {MAPPING mapping_name TO to_type}
  [EXCLUSIVE | NONEXCLUSIVE]
```

And finally, we create the multiply-typed domain by allowing a domain for an attribute in `CREATE TABLE`¹ command to be a set of possible types.

```
[attribute_domain | (type {, type})]
```

2.3 Hierarchy and interval mesodata types

For our example, we use hierarchy and interval types defined in the mesodata layer. We use the term hierarchy for the mesodata type rather than tree because we do not restrict nodes to having a single parent. We used the term *lattice* to describe our hierarchies in earlier work (Rice & Roddick 2000), but because

¹The choice we have made of extending the `CREATE DOMAIN` command then allowing an attribute to take values from more than one of these domains is possibly unfortunate, as it implies that an attribute can be drawn from different domains, rather than that a domain can be formed from values of different types.

they can include overlapping nodes, it is possible for two nodes not to have a unique least upper bound, so the term lattice is not general enough. Our structures are not as general as a DAG (directed acyclic graph), because they do have a defined *top* node (\top) and *bottom* node (\perp). The directedness of our hierarchy is also different from that of a DAG: you can traverse its edges in both directions, but there is a semantic difference between going towards \perp (more specific) and towards \top (more general).

The source relation for domains based on `interval` has the schema (DESCRIP, START, FINISH) with key DESCRIP. It is used to associate the end-points of the interval with the stored string used to represent it in the attribute, and is required when the RDBMS does not have a base interval type. START and FINISH can have any numeric or date/time type: the underlying logic is the same whether the intervals are numeric or temporal. Operations defined on intervals include the Allen relations (Allen 1983):

- $equals(i_1, i_2)$
- $before(i_1, i_2) / after(i_2, i_1)$
- $starts(i_1, i_2) / startedBy(i_2, i_1)$
- $during(i_1, i_2) / contains(i_2, i_1)$
- $finishes(i_1, i_2) / finishedBy(i_2, i_1)$
- $meets(i_1, i_2) / metBy(i_2, i_1)$
- $overlaps(i_1, i_2) / overlappedBy(i_2, i_1)$

In addition, we use $startOf(i_1)$ and $finishOf(i_1)$ to retrieve the end-points of interval i_1 .

The source relation for domains based on `hierarchy` describes the structure of the hierarchy: it has the schema (CHILD, PARENT) which has a key consisting of both attributes because a CHILD can have more than one PARENT. Required operations include predicates *childOf*, *parentOf*, *descendentOf*, *ancestorOf* and *inFamily*, and set-valued functions *allDescendents*, *allAncestors* and *family*. Let *SR* be the source relation, then we define:

- $childOf(x, y)$ is true if $(x, y) \in SR$.
- $parentOf(x, y)$ is true if $(y, x) \in SR$.
- $descendentOf(x, y)$ is true if $childOf(x, y) \vee (\exists(z) \wedge childOf(x, z) \wedge descendentOf(z, y))$.
- $ancestorOf(x, y)$ is true if $parentOf(x, y) \vee (\exists(z) \wedge parentOf(x, z) \wedge ancestorOf(z, y))$.
- $inFamily(x, y)$ is true if $x = y \vee descendentOf(x, y) \vee ancestorOf(x, y)$.
- $allDescendents(x)$ is the set $\{y | descendentOf(x, y)\}$.
- $allAncestors(x)$ is the set $\{y | ancestorOf(x, y)\}$.
- $family(x)$ is the set $\{y | inFamily(x, y)\}$.

These operations are part of the mesodata type and do not need to be coded by the user.

3 Multiply-typed domains

As an example of a multiply-typed domain, we introduce a data model for incomplete data where that incompleteness can be of three different base types (i.e. values with variable granularity, values that are vague, or where an interval is used to represent a single value). The same queries can be posed over attribute values of any of these types: the difference between them lies in the data structures used for each type and the corresponding operations used to answer the queries. Our data model uses a third truth value *unknown* and *hierarchical domains* to cope with the partial knowledge. In other work (Rice & Roddick 2000) we discuss an earlier version of this data model.

In this section we give an overview of this intelligent domain through an example based on archaeological data. In addition to the multiple types, we show how attributes with different semantics can use the same base values. This requires an implementation that allows the same stored domain values to be used with multiple semantics.

3.1 The example domain

We use in our example two relations KILNS shown in Table 1 and POTS shown in Table 2 containing information about pots and pottery kilns in Roman Britain (Swan 1984). The date values given for the attributes `inOperation` in KILNS and `dateCreated` in POTS are expressed in three different forms: using the names of Emperors, as an interval of years, or using vague terms such as *mid I* (i.e. in the middle of the first century). They reflect the terms used by the archaeologists who conducted the initial research over a period of nearly 200 years. All three forms use values which explicitly or implicitly define a range of values, but the semantics of the domain for each attribute are different – whereas `inOperation` defines the interval during which the kiln is believed to have been in operation, `dateCreated` uses the same values to represent not an interval, but an unknown point of time somewhere in that interval. The sorts of questions that archaeologists would like to answer about the pottery industry using these data include:

- What kilns were operating during the reign of Nero?
- Were the Hardingstone 1 and Binsted 15 kilns operating concurrently?
- Which kilns could have manufactured this pot?

To answer questions like these for the data shown, it is necessary to be able to map between the different representations used, which could be done by translating all dates into numerical intervals on data entry. However, this can lead to loss of information: if *Claudian* is translated into [41, 54], it loses the historical context of the original estimate, and if *early I* is changed to, say, [1, 30] the vagueness inherent in the original form disappears.

3.2 Hierarchical Domains

By hierarchical domains we mean domains where there is a hierarchical structure between (at least two of) the elements of the domain. The examples $D_1 = \{\text{Claudio-Neronian, pre-Flavian, Flavian, Claudian, Neronian}\}$, $D_2 = \{[50,100], [30,60], [70,100], [50,60]\}$ and $D_3 = \{\text{early-mid I, mid-late I, early I, mid I, late I}\}$ whose structures are shown in Figure 2 demonstrate this. Connections between nodes in the hierarchy represent a *containment relationship* – the lower of two connected nodes is (at least partially) contained within the upper node. We call the upper node of two connected nodes the *concept* and the lower node the *element*, following the usage introduced in (Roddick 1994). Any node in a hierarchy with both a child and a parent can be either a concept or an element, depending on which connection is being considered. An unlabelled top node \top is shown for each hierarchy, which by definition completely contains every node in the hierarchy. Nodes further down the hierarchy are more specific. For simplicity, the bottom node of the hierarchy \perp (which represents the empty set ϕ) is not shown. In the domains shown in Figure 2, the node *mid I* demonstrates the multiple-parent structure, because it belongs to (is contained in) both *early-mid I* and *mid-late I*.

There are three kinds of containment relationships shown in Figure 2 by the labels N , O and S . N is the ‘normal’ containment relationship where the element (lower node) is completely contained within the larger concept (upper node), O is the overlapping relationship where each node overlaps the other, and S is the relationship between two synonyms. For example, *pre-Flavian* and *Claudio-Neronian* are synonyms, and *early-mid I* and *mid-late I* are overlapping. The hierarchical structure has been retained in the presence of O and S containments by choosing one of the pair of concepts involved to be lower in the hierarchy than the other.² In both cases the choice is arbitrary, because each of the connected nodes contains the other, either partially (O containment) or completely (S containment). Although N , O and S containments can apply to sets in general, the domain elements in Figure 2 are all intervals, even if the bounds of the interval are not obvious (hierarchy (a)), or not precisely known (hierarchy (c)). Of course, the containment relationship between two numerically-expressed intervals such as [50, 60] and [50, 100] can be worked out directly from their end-points, without recourse to stored hierarchical information.

3.3 Queries

The development of a new type of intelligent domain usually entails extensions to query languages. In our example, we introduce some new syntax to SQL to cope with queries involving the attributes `dateCreated` and `inOperation`. For `inOperation`

²Note that the O containments shown are an extension of the usual meaning of hierarchy. It is perhaps better to describe O and S links as sibling links rather than parent-child ones. Containment relationships were added to this data model to reduce the complexity of algorithms for querying this data.

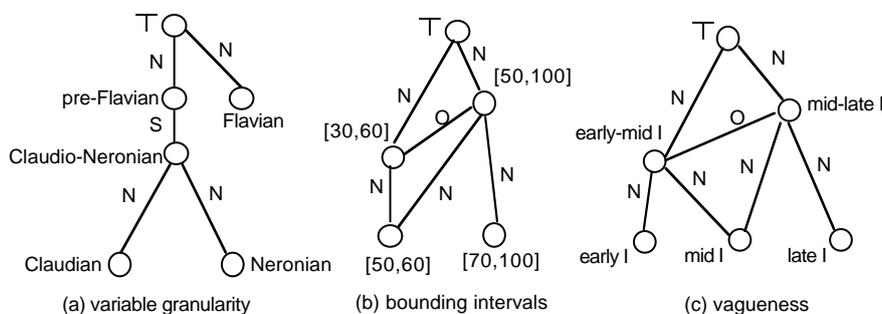


Figure 2: Domains showing connection types

Table 1: Relation KILNS

kiln	easting	northing	inOperation
Dates of varying granularity			
Hardingstone 1	476	257	pre-Flavian
Colchester 15	599	226	Claudio-Neronian
Chichester	486	105	Claudian
Stoke-on-Trent	387	343	Neronian
Biddlesden	464	240	Flavian
Dates as numerical intervals			
Oxford 5	455	206	[50, 100]
Binsted 15	477	141	[30, 60]
Hendon 1	517	194	[50, 60]
Dorchester 2	458	194	[70, 100]
Dates using vague terms			
Colchester 11	599	226	early-mid I
Little Houghton 5	481	260	mid-late I
Harrold	493	255	mid I
Kettering 1	489	278	late I

this syntax is based on Allen's logic for temporal intervals (Allen 1983). For example

```
inOperation CONTAINS 'Claudian'
```

The different semantics of `dateCreated` mean that conditions involving it are not comparing two intervals, however, but comparing a point *somewhere* in an interval with another interval. The Allen operations, as modified in (Vilain 1982) to compare a point with an interval, can be used here, but they need to be adapted to deal with uncertainty in the value of the point. Consider the condition

```
dateCreated DURING 'Claudian'
```

applied to a relation consisting of pots 1, 2 and 3 (see Table 2). This is *true* for pot 2, *false* for pot 3, and *unknown* for pot 1 (since Claudio-Neronian represents an interval that includes some dates that are Claudian and some that are not). On the other hand, there is no uncertainty about the condition

```
dateCreated DURING 'Claudio-Neronian'
```

which is *false* for pot 3, and *true* for the other pots. In hierarchies that only have *N* and *S* containment relationships, the condition `dateCreated DURING v` is *true* for all pots with a date that is a descendent of *v* in the hierarchy, but is *unknown* for all pots with a date that is an ancestor of *v* that is not synonymous

with *v* where *unknown* is a third truth value between *true* and *false* that indicates there is insufficient information to decide whether a condition evaluates to *true* or *false*. Truth tables for the operators \wedge (and), \vee (or) and \neg (not) are shown in Table 3.. The situation is more complex in hierarchies which include *O* containments: the truth value of the condition is also *unknown* for all descendents of *v* in the hierarchy to which there is no path that does not include an *O* containment.

In order to accommodate the *unknown* truth value, we introduce the keyword `MAYBE` as shown in these queries.

```
SELECT * FROM pots
WHERE dateCreated DURING 'Claudian';
```

```
SELECT * FROM pots
WHERE MAYBE dateCreated DURING 'Claudian';
```

```
SELECT * FROM pots
WHERE MAYBE dateCreated
NOT DURING 'Claudian';
```

`MAYBE` indicates we want to retrieve kilns for which the query condition is *true* or *unknown*. To answer these queries, we need to return all kilns where the condition `dateCreated DURING 'Claudian'` is *true*, *true* \vee *unknown*, and *unknown* \vee *false* respectively.

Table 2: Relation POTS

potID	description	easting	northing	dateCreated
Dates of varying granularity				
1	Belgic grey ware cooking pot	599	226	Claudio-Neronian
2	poppy-head beaker	486	105	Claudian
3	ring-necked flagon	464	240	Flavian
Dates as numerical intervals				
4	Clapham shelly ware bowl	517	194	[50, 60]
5	large storage jar	458	194	[70, 100]
Dates using vague terms				
6	narrow-mouthed bowl	493	255	mid I
7	lid-seated jar and lid	489	278	late I

Table 3: 3-valued logic truth tables

C	$\neg C$	$C_a \vee C_b$	t	u	f	$C_a \wedge C_b$	t	u	f
t	f	t	t	t	t	t	t	u	f
u	u	u	t	u	u	u	u	u	f
f	t	f	t	u	f	f	f	f	f

4 Implementing the domains

We have shown that, while the attributes `inOperation` and `dateCreated` both use the three domain hierarchies shown in Figure 2, their semantics differ. Different semantics for the same domain values can be accommodated by defining two domains on the same source relation, each with their own set of operations.

In this section, we develop different mesodata types for each of the three types of data for both these semantics, then create multiply-typed domains for each attribute using the mesodata types. It is a multi-layered approach, that allows re-use of the stored domain values.

The mesodata types we use in our example are:

interval: The `interval` type described above.

typedHierarchy: The `hierarchy` type described above extended to include an attribute specifying the containment relationship for each edge in the hierarchy.

pointInterval: The `interval` type where the interval is used to represent an unknown point somewhere within the interval.

pointTypedHierarchy: The `typedHierarchy` type where the nodes of the hierarchy represent points within an interval similar to the `pointInterval` type.

4.1 The extended mesodata types

First let us consider the source relations for the three hierarchies. The intervals shown in part (b) of Figure 2 can use the source relation schema described for the `interval` mesodata type in the previous section. For the other two hierarchies, we extend the schema for the source relation for `hierarchy` mesodata type to include the containment relationship

between `CHILD` and `PARENT` to be (`CHILD`, `PARENT`, `CONTAINMENT`).

Now consider the attribute `inOperation`. As discussed above, the attribute values are intervals, and query conditions use the Allen operations. For the interval hierarchy these need no adaptation (apart from perhaps interpreting `DURING` to also include `EQUALS`, `STARTS` and `FINISHES`). For the other two hierarchies though, these operations must be redefined using the structure of the hierarchy and the containment relationships of its edges. For example, consider the query condition `inOperation DURING d`. For the hierarchical domains, we can work out whether the query condition is *true* or *false* for a value v using these definitions:

- $\text{SYNONYMS}(d, v) = \text{true}$ if v is a synonym of d
- $\text{N_PATH}(d, v) = \text{true}$ if there is a normal path from d to v
- $\text{FULLDESCENDENT}(d, v) = \text{DESCENDENTOF}(d, v) \wedge \text{N_PATH}(d, v)$
- $\text{DURING}(d, v) = \text{SYNONYMS}(d, v) \vee \text{FULLDESCENDENT}(d, v)$

By a normal path we mean a path that does not contain an O containment.

The attribute `dateCreated`, however, represents a point located somewhere within the value used for the attribute, and query conditions use operations comparing a point with an interval, the keyword `MAYBE`, and three-valued logic as discussed earlier. For example, consider the query condition `dateCreated DURING d`. For the interval hierarchy we can use these definitions to determine the truth value of this condition for a value v :

- $\text{true} = \text{EQUALS}(d, v) \vee \text{CONTAINS}(d, v) \vee \text{STARTS}(d, v) \vee \text{FINISHES}(d, v)$
- $\text{unknown} = \text{DURING}(d, v) \vee \text{OVERLAPS}(d, v) \vee \text{OVERLAPPEDBY}(d, v) \vee \text{STARTEDBY}(d, v) \vee \text{FINISHEDBY}(d, v)$

Table 4: Source relations

Source Relation intervalsrel		
desc	start	finish
[50,60]	50	60
[70,100]	70	100
[30,60]	30	60
[50,100]	50	100
Source Relation emperorsrel		
child	parent	containment
Claudian	Claudio-Neronian	N
Neronian	Claudio-Neronian	N
Claudio-Neronian	pre-Flavian	S
Source Relation vaguerel		
child	parent	containment
early I	early-mid I	N
mid I	early-mid I	N
mid I	mid-late I	N
late I	mid-late I	N
early-mid I	mid-late I	O

- $false = \text{BEFORE}(d, v) \vee \text{AFTER}(d, v) \vee \text{MEETS}(d, v) \vee \text{METBY}(d, v)$

For the other two hierarchies, once again we need to consider domain structure and containments. We can determine the truth value of the condition for value v using these definitions:

- $true = \text{SYNONYMS}(d, v) \vee \text{FULLDESCENDENT}(d, v)$
- $unknown = \text{INFAMILY}(d, v) \wedge \neg(\text{SYNONYMS}(d, v) \vee \text{FULLDESCENDENT}(d, v))$
- $false = \neg\text{INFAMILY}(d, v)$

The operations not defined in this section are defined for the base mesodata types in Section 2. The difference in capitalisation is not meant to be significant.

4.2 Creating Mappings, Domains and Tables

To define the mappings between the source relations, we use the CREATE MAPPING command described earlier with a lookup table and a mapping function whose purposes are discussed below. The mappings required for our example are from Emperor names to intervals, and from vague terms to intervals.

```
CREATE MAPPING emperorMap
  LOOKUP emperorIntervals
  FUNCTION emperorToIntervals

CREATE MAPPING vagueMap
  LOOKUP vagueIntervals
  FUNCTION vagueToInterval
```

To create the domains for inOperation and dateCreated, we use the CREATE DOMAIN command described earlier. The AS clause defines the meso-data type to be used, the OF clause defines the base data type for the nodes in the hierarchy, and the OVER clause specifies the source relation to use.

```
CREATE DOMAIN inOpInterval
  AS interval OF CHAR(10)
  OVER intervalsrel
```

```
CREATE DOMAIN dateCrInterval
  AS pointInterval OF CHAR(10)
  OVER intervalsrel
```

```
CREATE DOMAIN inOpEmperor
  AS typedHierarchy OF CHAR(16)
  OVER emperorsrel
  MAPPING emperorMap TO inOpInterval
```

```
CREATE DOMAIN dateCrEmperor
  AS pointTypedHierarchy OF CHAR(16)
  OVER emperorsrel
  MAPPING emperorMap TO dateCrInterval
```

```
CREATE DOMAIN inOpVague
  AS typedHierarchy OF CHAR(12)
  OVER vaguerel
  MAPPING vagueMap TO inOpInterval
```

```
CREATE DOMAIN dateCrVague
  AS pointTypedHierarchy OF CHAR(12)
  OVER vaguerel
  MAPPING vagueMap TO dateCrInterval
```

We now use these domains to create tables KILNS and POTS. The syntax used for the domains of inOperation and dateCreated shows that values from each of the three domains listed can be used for that attribute.

```
CREATE TABLE kilns (
  kiln CHAR(20) PRIMARY KEY,
  easting INTEGER,
  northing INTEGER,
  inOperation (inOpInterval,
               inOpEmperor,
               inOpVague))
```

```
CREATE TABLE pots (
  potID INTEGER PRIMARY KEY,
  description CHAR(30),
  easting INTEGER,
  northing INTEGER,
  dateCreated (dateCrInterval,
               dateCrEmperor,
               dateCrVague))
```

Table 5: Mapping between Hierarchies and Intervals

Mapping Emperors to Intervals				
emperor	start		finish	
Claudian	41		54	
Neronian	54		68	
Flavian	68		96	
Claudio-Neronian	startOf(Claudian)		finishOf(Neronian)	
pre-Flavian	startOf(Claudio-Neronian)		finishOf(Claudio-Neronian)	
Mapping Vague Terms to Intervals				
vagueTerm	start	finish	startMin	finishMax
early I	1	33	1	startOf(mid I)
mid I	34	66	finishOf(early I)	startOf(late I)
late I	67	100	finishOf(mid I)	100
early-mid I	startOf(early I)	finishOf(mid I)		
mid-late I	startOf(mid I)	finishOf(late I)		

The source relations describing the hierarchical structure of the domains are shown in Table 4. It is not necessary to include any edges connecting nodes to \top , as these always have N containment.

Table 5 shows how to map the two hierarchical structures to intervals. This means providing start and finish values for the Emperor's reigns and the vague terms. In the case of the vague terms, minimum and maximum values have been given as well as default start and finish values to allow the default values to be varied if desired, but only in such a way that the relationships between the terms remain consistent. Wherever possible, `startOf` and `finishOf` operations are included in the table to reduce redundancy. The lookup table and mapping function defined in the `CREATE MAPPING` command are used to implement this mapping. The lookup table contains the numerical values in Table 5, and the mapping function is used to calculate the values using the `startOf` and `finishOf` operations.

4.3 Discussion of the Implementation

The implementation described in this paper comprises four distinct tasks:

Implementing the basic mesodata types.

The mesodata approach is new, and the only mesodata types implemented previously are graph, weighted graph, and circular list, so the hierarchy and interval mesodata types must be implemented. This task would not normally be part of the development of a database using a mesodata type.

Implementing the adapted mesodata types.

The `typedHierarchy`, `pointInterval` and `point-TypedHierarchy` implementations are based on those of interval and hierarchy. Once created, these are reusable mesodata types like any other, but the creation of a new intelligent domain may require adaptations of existing mesodata types like these.

Implementing the intelligent domain. This task involves the implementation of the mapping functions and different semantics for the query operations for the attributes `inOperation` and `dateCreated`.

Implementing the database This task incorporates the creation and population of the specific domains, mappings and relations.

These tasks are shown in decreasing order of likelihood of being required for a particular application.

The non-standard SQL syntax is handled by wrappers which perform any required mesodata operations and convert the various data definition and manipulation commands to standard SQL, as mesodata is still at the proof-of-concept stage.

In concept, a mesodata-style implementation of an intelligent domain is three-layered. The implemented algorithms are built using the operations defined for the mesodata types used in the domain, which in turn use the operations for the DBMS's base data types. In comparison, a direct implementation is two-layered: its special-purpose data structures and algorithms are built directly on top of the base data type, possibly enhancing its efficiency at the expense of re-use. As an experiment, our example domain is being implemented using both methods to see how they differ in ease of implementation, and efficiency of operation.

For illustrative purposes, consider the mesodata implementation of the query³

```
SELECT * FROM KILNS
WHERE inOperation DURING 'pre-Flavian'
```

for the `inOpEmperor` part of the multiply-typed `inOperation` domain. This requires identification of the tuples in the `KILNS` table with a value for `inOperation` which lies entirely within the pre-Flavian era. The algorithm `CALC_DURING` returns the set DV_{during} of domain values which satisfy this condition (in our example these are 'Claudian', 'Claudio-Neronian' and 'pre-Flavian') using as input d ('pre-Flavian', the domain value being matched) and DV_{all} (the set of `inOpEmperor` domain values used in `KILNS`). `CALC_DURING` uses the `SYNONYMS` and `FULLDESCENDENT` operations described in Section 4.1.

Algorithm `CALC_DURING`(d , DV_{all})

```
BEGIN
  Initialise  $DV_{during}$  to  $\phi$ 
```

³It will be necessary to introduce syntax to determine which of the multiple types the value 'pre-Flavian' belongs to, especially where there is more than one possible as is the case with our example.

```

FOR (each  $v \in DV_{all}$ )
  IF SYNONYMS( $d, v$ )  $\vee$  FULLDESCENDENT( $d, v$ )
    Add  $v$  to  $DV_{during}$ 
  ENDIF
ENDFOR
END

```

5 Conclusion and Further Research

Data integration often leads to compromise in the adoption of schemas that do not fit the data very well, in order to incorporate data from different sources into a single global schema. At the attribute level, this problem can be addressed by allowing attribute domains to accommodate multiple types. We have shown in this paper that the concept of mesodata can be used to define such domains.

We believe that the mesodata modelling methodology provides a handy tool for developing novel intelligent domains of all types. In particular, the ability to separate the domain values and structure from the semantics of attributes defined using the domain proved very useful for this data model, by providing a paradigm for thinking about the modelling process as well as enabling the reuse of the same set of complex values for attributes with different semantics.

Implementation of these ideas is already underway. The MySQL RDBMS (MySQL 2003) is being used for this purpose. We are implementing the same data model both with and without using mesodata techniques. Analysis of these algorithms so far shows no significant theoretical difference in complexity. Comparisons are being undertaken to determine whether there is a difference in practice.

6 Acknowledgements

The authors would like to thank the anonymous reviewers for their very helpful comments on our paper.

References

- Abraham, T. & Roddick, J. F. (1999), 'Survey of spatio-temporal databases', *GeoInformatica* **3**(1), 61–99.
- Allen, J. (1983), 'Maintaining knowledge about temporal intervals', *Communications of the ACM* **26**(11, November 1983), 832843.
- Connolly, T. & Begg, C. (2005), *Database Systems: A Practical Approach to Design, Implementation and Management, 4th Edition*, Addison Wesley.
- de Vries, D., Rice, S. & Roddick, J. F. (2004), In support of mesodata in database management systems, in 'DEXA 2004', Springer, Zaragoza, Spain.
- de Vries, D. & Roddick, J. F. (2004), Facilitating database attribute domain evolution using mesodata, in F. Grandi, ed., 'Third International Workshop on Evolution and Change in Data Management (ECDM2004)', Lecture Notes in Computer Science, Springer, Shanghai.
- Dey, D. & Sarkar, S. (1996), 'A probabilistic relational model and algebra', *ACM Transactions on Database Systems* **21**(3), 339369.
- Egenhofer, M. J. & Franzosa, R. D. (1991), 'Point-set topological spatial relations', *International Journal for Geographical Information Systems* **5**(2), 161–174.
- Lorentz, D. & Gregoire, J. (2003), *Oracle Database SQL Reference 10g Release 1 (10.1)*, Oracle Corporation.
- Melton, J. & Simon, A. R. (2002), *SQL:1999 – Understanding Relational Language Components*, Morgan Kaufmann Publishers.
- MySQL (2003), 'SQL open source software'.
- Rice, S. & Roddick, J. F. (2000), Lattice-structured domains, imperfect data and inductive queries, in M. Ibrahim, J. Kung & N. Revell, eds, '11th International Conference on Database and Expert Systems Applications, DEXA 2000', Lecture Notes in Computer Science, Springer, London, pp. 664–674.
- Roddick, J. (1994), A Model for Temporal Inductive Inference and Schema Evolution in Relational Database Systems, Doctor of philosophy, La Trobe University.
- Schneider, M. (1997), *Spatial Data Types for Database Systems*, Vol. 1288 of *Lecture Notes in Computer Science*, Springer.
- Snodgrass, R. T. (1995), *The TSQL2 Temporal Query Language*, Kluwer Academic Publishers.
- Swan, V. G. (1984), *The pottery kilns of Roman Britain*, Royal Commission for Historical Monuments.
- Vilain, M. B. (1982), A system for reasoning about time, in 'National Conference on Artificial Intelligence', Pittsburg, PA, pp. 197–201.
- Zeng, J. (1999), Research and practical experiences in the use of multiple data sources for enterprise-level planning and decision making: A literature review, Technical report, Center for Technology in Government, University at Albany / SUNY.

On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling*

Nick Russell¹Wil M.P. van der Aalst^{2,1}
Petia Wohed³Arthur H.M. ter Hofstede¹

¹School of Information Systems, Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia
{n.russell, a.terhofstede}@qut.edu.au

²Department of Technology Management, Eindhoven University of Technology
GPO Box 513, NL5600 MB Eindhoven, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl

³Department of Computer and Systems Sciences, Stockholm University/KTH
Forum 100, 164 40 Kista, Sweden
petia@dsv.su.se

Abstract

UML is posited as the “swiss army knife” for systems modelling and design activities. It embodies a number of modelling formalisms that have broad applicability in capturing both the static and dynamic aspects of software systems. One area of UML that has received particular attention is that of Activity Diagrams (ADs), which provide a high-level means of modelling dynamic system behaviour. In this paper we examine the suitability of UML 2.0 Activity Diagrams for business process modelling, using the Workflow Patterns as an evaluation framework. The Workflow Patterns are a collection of patterns developed for assessing control-flow, data and resource capabilities in the area of Process Aware Information Systems (PAIS). In doing so, we provide a comprehensive evaluation of the capabilities of UML 2.0 ADs, and their strengths and weaknesses when utilised for business process modelling.

1 Introduction

The Unified Modeling Language (UML) is increasingly being seen as the *de-facto* standard for software modelling and design. The most recent version (2.0) (OMG 2004) includes 13 distinct modelling notations ranging from high-level use case diagrams, which depict the interactions and relationships between (human) actors and major business functions, through to low-level object diagrams which capture instances of individual data objects, their constituent data elements and values, and their relationships with other data objects.

The various modelling notations essentially divide into three main groups:

- Behaviour diagrams, which describe the overall functionality of the software at a relatively high level of abstraction;
- Interaction diagrams, which further augment the behaviour diagrams and present a more detailed

description of system functionality in terms of object interactions; and

- Structure diagrams, which capture the underlying static structure of a software system at various levels from individual objects to overall application packages.

At its heart, UML is an object-oriented set of notations and is particularly effective for capturing detailed design models of software systems in a form which is suitable for translation into some form of enactment technology (usually program code) either by suitably qualified developers or in a semi-automated manner via CASE technology. However, the breadth of UML ensures that it also has potential applicability in a number of other scenarios such as *business process modelling* although in such a distinct domain, some notations (e.g. the class of behaviour diagrams) are likely to be more useful than others.

Over the past decade, the economics of software usage have begun to change, and it is increasingly common for new systems to be based on modifications of widely distributed software products rather than being custom software developments. There is also a broader view being taken as to the scope in which a system operates and the recognition that true cost-benefits can only occur when software processes are aligned with organisational processes. These notions have been reinforced by the advent of organisation-wide software packages such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM) and Workflow Management (WFM) systems which bind together multiple operational groups within an organisation in a set of integrated business processes.

As a consequence of this shift, there is an increased interest in software process modelling in an organisational context – generally termed *business process modelling* or *enterprise modelling*, depending on whether the focus of the modelling is on the business process or the overall organisation. Several modelling techniques have been proposed as an all-encompassing standard for this role, however no one formalism is predominant in this area (Mendling, Neumann & Nüttgens 2005). One of the major reasons cited for this (zur Muehlen & Rosemann 2004) is the wide disparity in the needs and viewpoints of various modellers and designers.

In this paper, we investigate the *suitability* of Activity Diagrams for business process modelling. This notation is the most detailed form of process modelling within UML. However, its applicability to the

*This work is funded in part by ARC Discovery Project DP0451092.

Copyright ©2006, Australian Computer Society, Inc. This paper appeared at Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Australia. Conferences in Research and Practice in Information Technology, Vol. 53. Markus Stumptner, Sven Hartmann and Yasushi Kiyoki, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

business process modelling domain in a general sense is not immediately evident and merits more detailed examination in order to determine what advantages it offers and what its shortcomings are.

We base this evaluation on the Workflow Patterns¹, a taxonomy of generic, recurring constructs originally devised to evaluate workflow systems, and more recently used to successfully evaluate workflow standards, business process languages and process-aware information systems in general (Dumas & ter Hofstede 2001, White 2004, Wohed, Perjons, Dumas & ter Hofstede 2003). In accordance with Jablonski and Bussler's original classification (Jablonski & Bussler 1996), these patterns span the control-flow, data and resource perspectives of PAIS. Our choice of this evaluation framework is based on the fact that it is 1) widely used, 2) well accepted, 3) comprehensible to the IT practitioner, 4) at a sufficiently detailed level of abstraction to provide a comprehensive basis for assessing their capabilities of business process modelling languages and 5) the most complete and powerful framework for this form of assessment currently in existence.

The main contributions of this paper are as follows:

- It is the first multi-perspective evaluation of the expressive capabilities of UML 2.0 ADs;
- It provides an assessment of the overall suitability of UML 2.0 ADs for business process modelling; and
- It identifies several areas of potential improvements to UML 2.0 ADs to further strengthen their use for this purpose.

This paper focuses on the the new version of UML Activity Diagrams (ADs) 2.0 (OMG 2004), which differ considerably from their precursor UML 1.4 ADs.² Previous evaluations (cf. (Dumas & ter Hofstede 2001, Opdahl & Henderson-Sellers 2002)) have analysed the expressive power of UML 1.4 ADs. There has also been a review of the capabilities of the control-flow perspective of UML 2.0 ADs (White 2004), however the limited focus of this investigation has restricted its usefulness as a means of assessing their overall suitability for general modelling purposes.

The remainder of this paper proceeds as follows: Sections 2 and 3 provide an overview of business process modelling languages and UML 2.0 ADs respectively. Sections 4, 5 and 6 present evaluations of the control-flow, data and resource perspectives of UML 2.0 ADs. Section 7 reviews the suitability of UML 2.0 ADs for business process modelling in light of the findings in the preceding sections. It also offers some recommendations for further improving their capabilities in this area.

2 Business Process Modelling Languages

Business process modelling is essentially a convergence of two related modelling domains: *process modelling* (cf. (Curtis, Kellner & Over 1992, Rolland 1997, Rolland 1998)) which seeks to provide "an abstract representation of a process architecture, design, or definition" ((Humphrey & Feiler 1992), p.33) and *enterprise modelling* or *business modelling* which focuses on documenting an organisation from a holistic standpoint, capturing details of its overall purpose and

¹See www.workflowpatterns.com for comprehensive details.

²The semantics of UML 2.0 ADs are based on token flow instead of statecharts as in UML 1.4.

goals in addition to more concrete details such as organisational structure and significant business activities (cf. (Vernadat 1996, Bubenko, Persson & Stirna 2001, Eriksson & Penker 2000, Marshall 1999)).

Whilst there is significant overlap between them, they are generally viewed as having distinct motivations (Jablonski & Bussler 1996), and this is best exemplified by the formalisms used for modelling in the two areas. Process models are usually based on a single technique, such as Data Flow Diagrams (DFDs), Event-driven Process Chains (EPCs), UML Activity Diagrams or Petri-Nets, which is used to capture the details of the process in question. In contrast, enterprise modelling generally requires a range of modelling techniques to be used in conjunction with each other in order to capture the required domain information. This tends to favour the use of integrated suites of modelling techniques such as ARIS (Scheer 2000), UML (Eriksson & Penker 2000, Marshall 1999) and EKD (Bubenko et al. 2001) which possess a sufficiently broad range of integrated modelling formalisms.

Business process modelling essentially seeks to provide a detailed description of a business process in an organisational context. There are a range of potential modelling languages that can be used for this purpose and Kueng *et. al.* (Kueng, Kawalek & Bichler 1996) have proposed a taxonomy of business process modelling techniques which classifies them into four groups:

- *Activity-oriented approaches* - focusing on the definition of business processes as a sequence of activities;
- *Object-oriented approaches* - leveraging the more comprehensive modelling constructs of object-orientation to capture business processes;
- *Role-oriented approaches* - modelling business processes based on the specific organisational roles and responsibilities involved; and
- *Speech-act approaches* - viewing business processes in the context of the speech-act language action paradigm.

Other considerations for the selection of an appropriate business process modelling language to use in a particular scenario include the *kind* of process being modelled and the *purpose* for which the modelling is being undertaken. Rolland (1998) classifies processes into three kinds: *strategic* - which investigate alternative ways of achieving a required outcome and produce a plan for doing so; *tactical* - which focus on the tactics for achieving the plan; and *implementation* - which describe how the plan will be achieved. Similarly, individual process models may be developed with one of three possible aims: *descriptive* - which describe what actually happens during a process; *prescriptive* - which define how a process might or should be performed; and *explanatory* - which detail the rationale for a process and link it to the requirements it must fulfill.

3 UML 2.0 Activity Diagrams

In UML Activity Diagrams the fundamental unit of behaviour specification is the Action. "An action takes a set of inputs and converts them to a set of outputs, though either or both sets may be empty" ((OMG 2004), p.229). Actions may also modify the state of the system. The language provides a very detailed action taxonomy, identifying more than 40 different action types in detail. A comprehensive

discussion of them is beyond the scope of this paper and in Figure 1a we only present the action types that we have found to be most relevant to our evaluations. These are the generic Action concept, Accept Event, Send Signal, and Call Behavior Action constructs.

Action/Activity 	AcceptEvent 	InitialNode 	ActivityFinal 	FlowFinal 	
CallBehaviorAction 	SendSignal 	Decision 	Merge 	Fork 	Join
a) Actions		b) Control Nodes			

Figure 1: The main constructs in UML 2.0 ADs

In order to represent the overall behaviour of a system, the concept of the Activity is used. Activities are composed of actions and/or other activities and they define dependencies between their elements. Graphically, they are composed of nodes and edges. The edges connect the nodes in sequential order. Nodes represent either Actions, Activities, Data Objects, or control nodes. The various types of control nodes are shown in Figure 1b.

4 The Control-Flow Perspective in UML 2.0 ADs

In this section we examine the control-flow perspective of UML 2.0 ADs and their ability to represent a series of twenty common control-flow modelling requirements that occur when defining process models. These requirements are described in terms of the Workflow Control Patterns (van der Aalst, ter Hofstede, Kiepuszewski & Barros 2003). The material in this section summarises the findings in (Wohed, van der Aalst, Dumas, ter Hofstede & Russell 2005) thus providing the basis for a comprehensive evaluation of UML 2.0 ADs from multiple perspectives.

4.1 Basic control patterns

Basic control-flow patterns define elementary aspects of process control. These are analogous to the definitions of elementary control-flow concepts laid down by the Workflow Management Coalition (WFMC 1999). There are five of these patterns:

- WCP1: *Sequence* – the ability to depict a sequence of activities;
- WCP2: *Parallel split* – the ability to capture a split in a single thread of control into multiple threads of control which can execute in parallel;
- WCP3: *Synchronisation* – the ability to capture a convergence of multiple parallel subprocesses/activities into a single thread of control thus synchronising multiple threads;
- WCP4: *Exclusive choice* – the ability to represent a decision point in a workflow process where one of several branches is chosen; and
- WCP5: *Simple merge* – the ability to depict a point in the workflow process where two or more alternative branches come together without synchronisation.

All five of these patterns can be captured by UML 2.0 ADs. The specific representation of each of these patterns is illustrated in Figure 2.

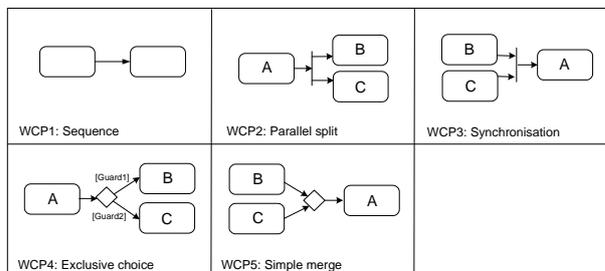


Figure 2: Basic control patterns in UML 2.0 ADs

4.2 Advanced branching & synchronisation patterns

This class of patterns corresponds to advanced branching and synchronisation scenarios that often do not have direct realisations in PAIS but are relatively common in real-life business processes. There are four of these patterns:

- WCP6: *Multiple choice* – the ability to represent a divergence of the thread of control into several parallel branches on a selective basis;
- WCP7: *Synchronising merge* – the ability to depict the synchronised convergence of two or more alternative branches;
- WCP8: *Multiple merge* – the ability to represent the unsynchronised convergence of two or more distinct branches. If more than one branch is active, the activity following the merge is started for every activation of every incoming branch; and
- WCP9: *Discriminator* – the ability to depict the convergence of two or more branches such that the first activation of an incoming branch results in the subsequent activity being triggered and subsequent activations of remaining incoming branches are ignored.

The multiple choice, multiple merge and discriminator patterns can be captured directly in UML 2.0 ADs and they are illustrated in Figure 3. The synchronising merge pattern cannot be directly modelled.

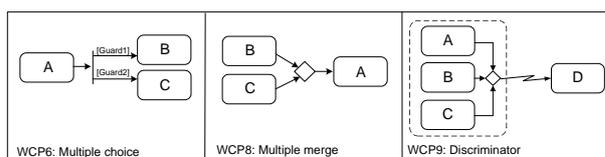


Figure 3: Advanced branching & synchronisation patterns in UML 2.0 ADs

4.3 Structural patterns

Structural patterns identify whether the modelling formalism has any restrictions in regard to the way in which processes can be structured (particularly in terms of the type of loops supported and whether a single terminating node is necessary). There are two of these patterns:

- WCP10: *Arbitrary cycles* – the ability to represent loops in a process that have multiple entry or exit points; and
- WCP11: *Implicit termination* – the ability to depict the notion that a given subprocess should be terminated when there are no remaining activities to be completed (i.e. no explicit unique termination node is needed).

Both of these patterns are directly supported in UML 2.0 ADs.

4.4 Multiple instance patterns

This class of patterns relates to situations where there can be more than one instance of an activity active at the same time for the same process instance. There are four of these patterns:

- WCP12: *MI without synchronisation* – the ability to initiate multiple instances of an activity within a given process instance;
- WCP13: *MI with a priori design time knowledge* – the ability to initiate multiple instances of an activity within a given process instance. The number of instances is known at design time. Once all instances have completed, a subsequent activity is initiated;
- WCP14: *MI with a priori runtime knowledge* – the ability to initiate multiple instances of an activity within a given process instance. The number of instances varies but is known at runtime before the instances must be created. Once all instances have completed, a subsequent activity is initiated; and
- WCP15: *MI without a priori runtime knowledge* – the ability to initiate multiple instances of an activity within a given process instance. The number of instances varies but is not known at design time or at runtime before the instances must be created. Once all instances have completed, a subsequent activity is initiated. New instances can be created even while other instances are executing or have already completed.

The first three of these patterns can be captured in UML 2.0 ADs as illustrated in Figure 4. The MI without a priori runtime knowledge pattern is not directly supported in UML 2.0 ADs.

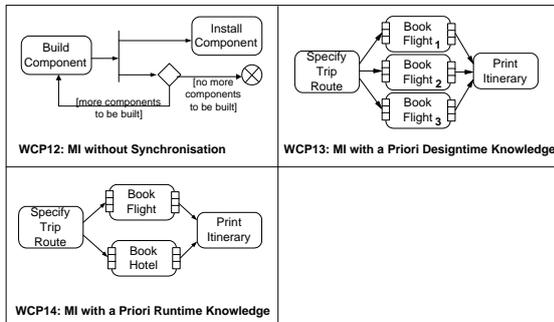


Figure 4: Multiple instance patterns in UML 2.0 ADs

4.5 State-based patterns

This class of patterns characterise scenarios in a process where subsequent execution is determined by the state of the process instance. There are three such patterns:

- WCP16: *Deferred choice* – the ability to depict a divergence point in a process where one of several possible branches should be activated. The actual decision on which branch is activated is made by the environment and is deferred to the latest possible moment;
- WCP17: *Interleaved parallel routing* – the ability to depict a set of activities that can be executed in arbitrary order; and

- WCP18: *Milestone* – the ability to depict that a specified activity cannot be commenced until some nominated state is reached which has not expired yet.

Owing to the absence of the notion of state, only the deferred choice pattern can be captured in UML 2.0 ADs. This is illustrated in Figure 5.

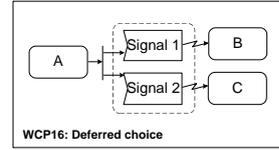


Figure 5: Deferred choice pattern in UML 2.0 ADs

4.6 Cancellation patterns

Cancellation patterns characterise the ability of the modelling formalism to represent the potential termination of activities and process instances in certain (specified) circumstances. There are two such patterns:

- WCP19: *Cancel activity* – the ability to depict that an enabled activity should be disabled in some nominated circumstance; and
- WCP20: *Cancel case* – the ability to represent the cancellation of an entire process instance (i.e. all activities relating to the process instance) in some nominated circumstance.

Both of these patterns can be captured in UML 2.0 ADs. The first is illustrated in Figure 6, the second is captured via the ActivityFinalNode construct.

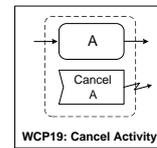


Figure 6: Cancel activity pattern in UML 2.0 ADs

5 The Data Perspective in UML 2.0 ADs

Extensions (Russell, ter Hofstede, Edmond & van der Aalst 2005) to the *Workflow Patterns Initiative* have focused on identifying and defining generic constructs that occur in the data perspective of PAIS. In total forty data patterns have been delineated in four distinct groups – data visibility, data interaction, data transfer and data-based routing. In this section, an analysis of UML 2.0 ADs is presented using the data patterns described in (Russell, ter Hofstede, Edmond & van der Aalst 2005).

5.1 Data visibility patterns

Data visibility patterns seek to characterise the various ways in which data elements can be defined and utilised within the context of a process. In general, this is determined by the main construct to which the data element is bound as it implies a particular scope in which the data element is visible and capable of being utilised. There are eight patterns which relate to data visibility:

- WDP1: *Task data* – data elements defined and accessible in the context of individual execution instances of a task or activity;

Nr	Pattern	2.0	Nr	Pattern	2.0
	Basic Control			Multiple Instance	
1	Sequence	+	12	MI without Synchronization	+
2	Parallel Split	+	13	MI with a priori Design Time Knowledge	+
3	Synchronisation	+	14	MI with a priori Runtime Knowledge	+
4	Exclusive Choice	+	15	MI without a priori Runtime Knowledge	-
5	Simple Merge	+		State-based	
	Adv. Branching & Synchronisation		16	Deferred Choice	+
6	Multiple Choice	+	17	Interleaved Parallel Routing	-
7	Synchronising Merge	-	18	Milestone	-
8	Multiple Merge	+		Cancellation	
9	Discriminator	+	19	Cancel Activity	+
	Structural		20	Cancel Case	+
10	Arbitrary Cycles	+			
11	Implicit Termination	+			

Table 1: Support for Control-Flow Patterns in UML 2.0 ADs

- WDP2: *Block data* – data elements defined by block tasks (i.e. tasks which can be described in terms of a corresponding decomposition) and accessible to the block task and all corresponding components within the associated decomposition;
- WDP3: *Scope data* – data elements bound to a subset of the tasks in a process instance;
- WDP4: *Multiple instance data* – data elements specific to a single execution instance of a task (where the task is able to be executed multiple times);
- WDP5: *Case data* – data elements specific to a process instance which are accessible to all components of the process instance during execution;
- WDP6: *Folder data* – data elements bound to a subset of the tasks in a process definition but accessible to all task instances regardless of the case to which they correspond;
- WDP7: *Workflow data* – data elements accessible to all components in all cases; and
- WDP8: *Environment data* – data elements defined in the operational environment which can be accessed by process elements.

In the case of UML 2.0 ADs, there is support for several of these patterns. The smallest operational unit in the context of these diagrams is the action. This corresponds to the notion of a process element or task and although the notion of task data is not directly supported, there is indirect support in the situation where a local action language is utilised which provides action-specific variables (see (OMG 2004), p.338-9).

Activities serve as the main grouping mechanism in UML 2.0 ADs and they have similar characteristics to the block construct in process definitions. The block data pattern is directly supported through parameters to activities (see (OMG 2004), p.363) which are accessible to all activity components. The concept of attributes (see (OMG 2004), p.341-7) is also provided which allows data elements to be defined which are scoped to a specific activity.

Scope data is not supported. The ActivityGroup construct (see (OMG 2004), p.359) seems to offer something analogous however the semantics of the construct are not defined.

Multiple instance data is directly supported and there are three situations where multiple instances of a given task may arise:

1. Where a task is specifically designated as having multiple instances in the process model – this facility seems to be provided by the ExpansionKind construct (see (OMG 2004), p.394) where the parallel option is chosen forcing parallel execution;
2. Where a task can be triggered multiple times e.g. it is part of a loop. This situation is allowable in UML 2.0 ADs (see (OMG 2004), p.345); and
3. Where two tasks share the same decomposition. This is also supported in UML 2.0 ADs as each activity decomposition is distinct and has a different set of tokens supplied to it at initiation (see (OMG 2004), p.360-1).

Case and folder data are not supported as there does not appear to be the notion of distinct execution instances in UML 2.0 ADs, rather all instances of a process model execute in the same context and are differentiated by distinct sets of control and object tokens flowing through the diagram. Workflow data is directly supported through data objects or Object-Nodes (see (OMG 2004), p.422) which are potentially accessible to all of the components in a UML 2.0 ADs. There does not appear to be the ability within UML 2.0 ADs to refer to data outside of the context of the diagram, and hence environment data is not supported.

5.2 Data interaction patterns

Data interaction patterns deal with the various ways in which data elements can be passed between components within a process instance and also with the operating environment (e.g. data transfer between a component of a process and an application, data store or interface that is external to the process). They examine how the characteristics of the individual components can influence the manner in which the trafficking of data elements occurs.

There are six internal data interaction patterns:

- WDP9: *Data elements flowing between task instances;*
- WDP10: *Data elements flowing to a block;*
- WDP11: *Data elements flowing from a block;*
- WDP12: *Data elements flowing to a multiple instance task instance;*
- WDP13: *Data elements flowing from a multiple instance task instance;* and

- WDP14: *Data elements flowing between process instances or cases.*

Data interaction between tasks is directly supported in UML 2.0 ADs by the `ObjectNode` construct (see (OMG 2004), p.422) which is the standard means of communicating data elements between activities.

Data interaction between block tasks and their decompositions has a similar analogy in UML 2.0 ADs in the form of data passing to and from activities. The standard means of doing this is via parameters (see (OMG 2004), p.363). Both the data interaction block task to sub-workflow and data interaction sub-workflow to block task patterns (WDP10 and WDP11) are directly supported.

Data interaction to and from multiple instance tasks has a direct analogy in UML 2.0 ADs in the `ExpansionRegion` construct (see (OMG 2004), p.395) which allows nominated regions of a process model to be executed multiple times in parallel (providing the `ExpansionKind` mode is set to parallel). Data passing into and out of the `ExpansionRegion` occurs using `ExpansionNodes` which provide the ability to map distinct sections of the input data set to specific execution instances and similarly completing instances can map their output to a specific section of the output data set. Hence the data interaction to and from multiple instance task patterns (WDP12 and WDP13) are directly supported.

There is no notion of distinct execution cases in UML 2.0 ADs, hence the data interaction – case to case pattern (WDP14) is not supported.

There are 12 external data interaction patterns, characterised by three dimensions:

- The type of process element – task, case or complete process – that is interacting with the environment;
- Whether the interaction is push or pull-based; and
- Whether the interaction is initiated by the process component or the environment.

Difficulties arise when examining UML 2.0 ADs in the context of this class of patterns as the UML approach assumes that an Activity Diagram represents the complete universe of discourse and does not provide the ability to reference or interact with elements that are external to it.

5.3 Data transfer patterns

Data transfer patterns focus on the way in which data elements are actually transferred between one process element and another. They aim to capture the various mechanisms by which data elements can be passed across the interface of a process element. There are seven distinct patterns in this category:

- WDP27: *Data transfer by value – incoming* – incoming data elements passed by value;
- WDP28: *Data transfer by value – outgoing* – outgoing data elements passed by value;
- WDP29: *Data transfer – copy in/copy out* – where a process element synchronises data elements with an external data source at commencement and completion;
- WDP30: *Data transfer by reference – without lock* – data elements are communicated between components via a reference to a data element in some mutually accessible location. No concurrency restrictions are implied;

- WDP31: *Data transfer by reference – with lock* – similar to WDP30 except that concurrency restrictions are implied with the receiving component receiving the privilege of read-only or dedicated access to the data element;
- WDP32: *Data transformation – input* – where a transformation function is applied to a data element prior to it being passed to a subsequent component; and
- WDP33: *Data transformation – output* – where a transformation function is applied to a data element prior to it being passed from a previous component.

In the context of UML 2.0 ADs, only three of these patterns are supported: WDP31: data transfer by reference – with lock - is the standard means of passing data elements into an activity as parameters. As UML 2.0 ADs adopt a token-oriented approach to data passing, these parameters – which typically relate to objects – are effectively consumed at activity commencement and only become visible and accessible to other activities once the specific activity to which they were passed has completed and returned them; WDP32: data transformation - input – can be achieved through the `ObjectFlow` transformation behaviour (see (OMG 2004), p.418) which allows transformation functions to be applied to data tokens as they are passed along connecting edges between activities; and WDP33: data transformation - output – as for pattern WDP32.

5.4 Data-based routing patterns

Data-based routing patterns capture the various ways in which data elements can interact with other perspectives and influence the overall execution of the process. There are seven (relatively self-explanatory) patterns in this category:

- WDP34: *Task precondition – data existence*;
- WDP35: *Task precondition – data value*;
- WDP36: *Task postcondition – data existence*;
- WDP37: *Task postcondition – data value*;
- WDP38: *Event-based task trigger*;
- WDP39: *Data-based task trigger*; and
- WDP40: *Data-based routing*.

The majority of these patterns are supported in UML 2.0 ADs. Both action and activity constructs include local preconditions and postconditions based on logical expressions (which may include data elements) framed in OCL (see (OMG 2004), p.336 and p.346). As a consequence, all of the task pre and postcondition patterns (WDP34 - WDP37) are directly supported. The `AcceptEventAction` construct (see (OMG 2004), p.334) provides direct support for the event-based task triggering pattern. Similarly, there is direct support for data-based routing via the `DecisionNode` construct and guard conditions on `ActivityEdges` (see (OMG 2004), p.387 and p.351). The lack of support for persistent state management within UML 2.0 ADs means that the data-based task trigger pattern cannot be captured.

Nr	Pattern		Nr	Pattern	
	Data Visibility			Data Interaction (Ext.) (cont.)	
1	Task Data	+/-	21	Env. to Case – Push-Oriented	–
2	Block Data	+	22	Case to Env. – Pull-Oriented	–
3	Scope Data	–	23	Workflow to Env. – Push-Orient.	–
4	Multiple Instance Data	+	24	Env. to Workflow – Pull-Orient.	–
5	Case Data	–	25	Env. to Workflow – Push-Orient.	–
6	Folder Data	–	26	Workflow to Env. – Pull-Orient.	–
7	Workflow Data	+		Data Transfer	
8	Environment Data	–	27	by Value – Incoming	–
	Data Interaction (Internal)		28	by Value – Outgoing	–
9	between Tasks	+	29	Copy In/Copy Out	–
10	Block Task to Sub-workflow Decomp.	+	30	by Reference – Unlocked	–
11	Sub-workflow Decomp. to Block Task	+	31	by Reference – Locked	+
12	to Multiple Instance Task	+	32	Data Transformation – Input	+
13	from Multiple Instance Task	+	33	Data Transformation – Output	+
14	Case to Case	–		Data-based Routing	
	Data Interaction (External)		34	Task Precondition – Data Exist.	+
15	Task to Env. – Push-Oriented	–	35	Task Precondition – Data Val.	+
16	Env. to Task – Pull-Oriented	–	36	Task Postcondition – Data Exist.	+
17	Env. to Task – Push-Oriented	–	37	Task Postcondition – Data Val.	+
18	Task to Env. – Pull-Oriented	–	38	Event-based Task Trigger	+
19	Case to Env. – Push-Oriented	–	39	Data-based Task Trigger	–
20	Env. to Case – Pull-Oriented	–	40	Data-based Routing	+

Table 2: Support for Data Routing Patterns in UML 2.0 ADs

6 The Resource Perspective in UML 2.0 ADs

Recent work (Russell, van der Aalst, ter Hofstede & Edmond. 2005) has focused on the resource perspective and the manner in which work is distributed amongst and managed by the resources associated with a business process. Our investigations have indicated that these patterns are relevant to all forms of PAIS including modelling languages such as XPDL and business process enactment languages such as BPEL4WS. In this section, we examine the resource perspective of UML 2.0 ADs and their expressive power in regard to work distribution.

Forty three workflow resource patterns have been identified in seven distinct groups:

- *Creation patterns* – which correspond to restrictions on the manner in which specific work items can be advertised, allocated and executed by resources;
- *Push patterns* – which describe situations where a PAIS proactively offers or allocates work to resources;
- *Pull patterns* – which characterise scenarios in which resources initiate the identification of work that they are able to undertake and commit to its execution;
- *Detour patterns* – which describe deviations from the normal sequence of state transitions associated with a business process either at the instigation of a resource or the PAIS;
- *Auto-start patterns* – which relate to situations where the execution of work is triggered by specific events or state transitions in the business process;
- *Visibility patterns* – which describe the ability of resources to view the status of work within the PAIS; and
- *Multiple resource patterns* – which describe scenarios where there is a many-to-many relationship between specific work items and the resources undertaking those work items.

In UML 2.0 ADs, the association of a particular action or set of actions with a specific resource is illustrated through the use of the ActivityPartition construct (see (OMG 2004), p.367). This may take many forms although the “swimlanes” notation is probably the most widely adopted means of presentation, where each lane indicates the resource that will be responsible for executing a specific subset (i.e. a partition) of the actions within an activity. Each partition has a name that corresponds to a specific resource or a group of resources to which the contained actions should be allocated at run-time. Partitions may be specified in four distinct ways: Classifier, Instance, Part, and Attribute and Value. The first two of these schemes (i.e. Classifier and Instance) are relevant in the context of resource allocation.

The direct allocation pattern (WRP1) is directly supported in UML 2.0 ADs as the ability to base a partition on a specific instance allows the actions to be associated with a single specified resource. There is no direct notion of roles within UML 2.0 ADs, although where a partition is based on a classifier, it is possible that the contained actions are allocated to multiple objects corresponding to the classifier (i.e. multiple resources). This is analogous to the notion of group allocation within a traditional workflow system. It is up to the individual resources to determine whether one or all of them will execute the assigned actions (see (OMG 2004), p.368-70). Consequently the requirements of the role-based allocation pattern (WRP2) are fully met and the pattern is directly supported. It is not necessary that actions within an activity belong to a partition and have a corresponding resource association, therefore the automatic execution pattern (WRP11) is also directly supported.

None of the other Creation Patterns are supported within UML 2.0 ADs. In the main, this is a consequence of the restrictive manner in which partitions are specified and the lack of support for relationships between distinct partitions. The attribute and value partition specifier seems to offer a means of implementing the deferred allocation pattern (WRP3) by delaying the need to identify a potential resource until run-time, however it is not possible to specify alter-

nate (i.e. parallel) courses of action based on different values of an attribute and even if it were, the necessity to enumerate a distinct course of action for each value (i.e. each potential resource) would make this approach unwieldy. Lack of an integrated authorisation framework, organisational model or access to an execution history also rules out any form of support for the authorisation (WRP4), organisational allocation (WRP9) and history-based allocation (WRP8) patterns respectively.

The execution semantics of a UML 2.0 AD are based on Petri Net token flow, hence actions become “live” once they receive a control-flow token. The resource associated with a given partition can have multiple actions executing (possibly in different partitions) at the same time. There is no notion of scheduling work execution or of resources selecting the work (i.e. the actions) they wish to undertake, hence there is minimal support for the Push, Auto-start or Multiple Resource patterns within UML 2.0 ADs. The following patterns from these classes are directly supported:

- WRP14: *Distribution by allocation - single resource* – the resource(s) associated with a given partition is immediately allocated an action once it is triggered;
- WRP19: *Distribution on enablement* – all actions in a partition are associated with the resource responsible for the partition when they are triggered;
- WRP36: *Commencement on creation* – an action is assumed to be live as soon as it receives a control-flow token;
- WRP39: *Chained execution* – once an action is completed, subsequent actions receive a control-flow token and are triggered immediately; and
- WRP42: *Simultaneous execution* – there are no constraints on how many partitions a given resource can be specified for or how many instances of these can be active at any one time.

None of the Pull, Detour or Visibility patterns are supported.

7 Conclusions

The pattern evaluations described in this paper indicate that whilst UML 2.0 ADs have merit as a notation for business process modelling, they are not suitable for all aspects of this type of modelling. They offer comprehensive support for the control-flow and data perspectives allowing the majority of the constructs encountered when analysing these perspectives to be directly captured. However, their suitability for modelling resource-related or organisational aspects of business processes is extremely limited. It is interesting to note that they are not able to capture many of the natural constructs encountered in business processes such as cases and the notion of interaction with the operational environment in which the process functions. These are limitations that they share with most other business process modelling formalisms and reflect the overwhelming emphasis that has been placed on the control-flow and data perspectives in contemporary modelling notations.

The level of support observed for control-flow patterns (see Table 1 for a complete summary³) illustrates that there is relatively broad support for capturing the various types of control-flow constructs

³A “+” in the table indicates *direct support* for the pattern (i.e. there is a construct in the language that directly supports the pattern).

that may arise in actual business processes. In terms of addressing the patterns that are not directly supported, we would like to make the following recommendations:

- Given the difficulties in capturing state-based patterns, most notably the interleaved parallel routing pattern and the milestone pattern, it may be worthwhile providing direct support for the notion of the place as it exists in Petri nets. Petri net places capture the notion of “waiting state” in a much less restrictive way than the AcceptEventAction construct does;
- UML 2.0 ADs currently do not support the creation of new instances of an activity while other instances of that activity are already running. This could be resolved through extensions to the ExpansionRegion construct to allow further instances to be dynamically created after the activity has started; and
- Given the lack of support for the synchronising merge, a concept similar to the OR-join could be added to UML 2.0 ADs.

The data patterns evaluation is summarised in Table 2. This shows that the data perspective is also well supported. Furthermore, the following remarks can be made:

- There is no notion of cases or distinct process instances in UML 2.0 ADs, hence all data is effectively block-scoped by default and parallel threads of execution occur in the same data space. This could lead to some problematic situations when modelling highly data intensive and/or highly concurrent processes;
- The use of “tokens” as the fundamental underpinning for control and data flow introduces some subtle variations that do not exist in other PAIS (except those based on Petri-nets) – in particular data elements are truly consumed (and cease to exist) when they are passed to an activity for the duration of the activity. This also makes it difficult to actually share a data element/object between concurrent activities. On the other hand, it minimises concurrency problems;
- The token approach provides an effective basis for internal data interaction (and hence all patterns are “+”). In particular, multiple instance data handling seems to be supported for all three multiple instance situations: designated multiple instance tasks, multiply triggered tasks (loops) and block tasks with a common decomposition; and
- There does not seem to be any ability to model things “outside of the model” i.e. in the external environment. Hence there is no real ability to support external data interaction patterns. This may be addressed by using UML ADs in conjunction with other diagrams such as UML interaction, overview and sequence diagrams, but this requires that the relationships between these diagrams be carefully established.

The resource patterns evaluation is summarised in Table 3. As discussed, it indicates that the support in UML 2.0 ADs for the modelling of work distribution directives is relatively minimal. This reinforces the fact that UML 2.0 ADs tend to be control-flow and data-centric and mainly aim to capture simple static routing directives associated with actions. They do not provide a means of representing the subtleties associated with selective work allocation across a range

Nr	Pattern		Nr	Pattern	
	Creation Patterns			Pull Patterns (cont.)	
1	Direct Allocation	+	24	System-Determ. Wk Queue Cont.	-
2	Role-Based Allocation	+	25	Resource-Determ. Wk Queue Cont.	-
3	Deferred Allocation	-	26	Selection Autonomy	-
4	Authorisation	-		Detour Patterns	
5	Separation of Duties	-	27	Delegation	-
6	Case Handling	-	28	Escalation	-
7	Retain Familiar	-	29	Deallocation	-
8	Capability-Based Allocation	-	30	Stateful Reallocation	-
9	History-Based Allocation	-	31	Stateless Reallocation	-
10	Organisational Allocation	-	32	Suspension/Resumption	-
11	Automatic Execution	+	33	Skip	-
	Push Patterns		34	Redo	-
12	Distrib. by Offer - Single Resource	-	35	Pre-Do	-
13	Distrib. by Offer - Multiple Resources	-		Auto-Start Patterns	
14	Distrib. by Allocation - Single Resource	+	36	Commencement on Creation	+
15	Random Allocation	-	37	Creation on Allocation	-
16	Round Robin Allocation	-	38	Piled Execution	-
17	Shortest Queue	-	39	Chained Execution	+
18	Early Distribution	-		Visibility Patterns	
19	Distribution on Enablement	+	40	Conf. Unalloc. Work Item Visibility	-
20	Late Distribution	-	41	Conf. Alloc. Work Item Visibility	-
	Pull Patterns			Multiple Resource Patterns	
21	Resource-Init. Allocation	-	42	Simultaneous Execution	+
22	Resource-Init. Exec. - Alloc. Wk Items	-	43	Additional Resource	-
23	Resource-Init. Exec. - Offer. Wk Items	-			

Table 3: Support for Resource Patterns in UML 2.0 ADs

of possible resources and the management of those work items at run-time. In particular, there is no real support for modelling any form of work distribution other than direct allocation or role-based allocation. There is no opportunity to utilise data resources (either within the model or externally from the environment) thus any opportunity for modelling organisational, history-based or capability-based allocation is obviated. Similarly, there is no support for specifying any form of work distribution algorithm or employing varying styles of work distribution (e.g. push vs pull, offer vs allocation).

Other observations arising from the resource patterns analysis include:

- The fact that the partitions can result in actions being simultaneously allocated to more than one resource can lead to difficulties where a means of providing role-based work allocation to a single resource is required. It is important to note that the resolution of this situation must be addressed as part of the implementation of the actions within the Activity Diagram; and
- The ability to use OCL statements in the specification of partitions (and also for specifying relationships between partitions) would enhance the capability of UML 2.0 ADs to capture possible resource allocations, both in terms of precision and the range of work allocation strategies that could be represented.

References

- Bubenko, J., Persson, A. & Stirna, J. (2001), EKD user guide, Technical report, Royal Institute of Technology (KTH) and Stockholm University, Stockholm, Sweden.
- Curtis, B., Kellner, M. & Over, J. (1992), 'Process modelling', *Communications of the ACM* **35**(9), 75–90.
- Dumas, M. & ter Hofstede, A. (2001), UML activity diagrams as a workflow specification language, in M. Gogolla & C. Kobryn, eds, 'Proceedings of the Fourth International Conference on the Unified Modeling Language (UML 2001)', LNCS 2185, Springer, Toronto, Canada, pp. 76–90.
- Eriksson, H. & Penker, M. (2000), *Business Modeling with UML*, OMG Press, New York.
- Humphrey, W. & Feiler, P. H. (1992), Software process development and enactment: Concepts and definitions, Technical Report SEI-92-TR-4, SEI Institute, Pittsburgh, USA.
- Jablonski, S. & Bussler, C. (1996), *Workflow Management: Modeling Concepts, Architecture and Implementation*, Thomson Computer Press, London, UK.
- Kueng, P., Kawalek, P. & Bichler, P. (1996), How to compose an object-oriented business process model?, in S. Brinkkemper, K. Lytinen & R. Welke, eds, 'Proceedings of the IFIP WG8.1/WG8.2 Working Conference', Atlanta, GA, USA.
- Marshall, C. (1999), *Enterprise Modeling with UML*, Addison Wesley, Reading.
- Mendling, J., Neumann, G. & Nüttgens, M. (2005), A comparison of XML interchange formats for business process modelling, in L. Fischer, ed., 'Workflow Handbook 2005', Workflow Management Coalition, Lighthouse Point, Florida, USA, pp. 185–198.
- OMG (2004), UML 2.0 superstructure specification, Technical report. <http://www.omg.org/cgi-bin/doc?ptc/2004-10-02>.
- Opdahl, A. & Henderson-Sellers, B. (2002), 'Ontological evaluation of the UML using the Bunge-Wand-Weber model', *Software and System Modeling* **1**(1), 43–67.

- Rolland, C. (1997), A primer for method engineering, *in* 'Proceedings of the INformatique des ORganisations et Systèmes d'Information et de Décision (INFORSID'97)', Toulouse, France.
- Rolland, C. (1998), A comprehensive view of process engineering, *in* B. Pernici & C. Thanos, eds, 'Proceedings of the 10th International Conference on Advanced Information Systems Engineering (CAiSE'98)', Vol. 1413 of *Lecture Notes in Computer Science*, Springer, Pisa, Italy.
- Russell, N., ter Hofstede, A., Edmond, D. & van der Aalst, W. (2005), Workflow data patterns: Identification, representation and tool support, *in* 'Proceedings of the 25th International Conference on Conceptual Modeling (ER'2005)', Springer, Klagenfurt, Austria.
- Russell, N., van der Aalst, W., ter Hofstede, A. & Edmond, D. (2005), Workflow resource patterns: Identification, representation and tool support., *in* O. Pastor & J. Falcao é Cunha, eds, 'Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)', Vol. 3520 of *Lecture Notes in Computer Science*, Springer, Porto, Portugal, pp. 216–232.
- Scheer, A.-W. (2000), *ARIS - Business Process Modelling*, Springer, Berlin, Germany.
- van der Aalst, W., ter Hofstede, A., Kiepuszewski, B. & Barros, A. (2003), 'Workflow patterns', *Distributed and Parallel Databases* **14**(3), 5–51.
- Vernadat, F. (1996), *Enterprise Modeling and Integration*, Chapman and Hall, London.
- WFMC (1999), Workflow management coalition terminology and glossary, document status - issue 3.0, Technical Report WFMC-TC-1011, Workflow Management Coalition. http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf.
- White, S. (2004), Process modeling notations and workflow patterns, *in* L. Fischer, ed., 'Workflow Handbook 2004', Future Strategies Inc., Lighthouse Point, FL, USA., pp. 265–294.
- Wohed, P., Perjons, E., Dumas, M. & ter Hofstede, A. (2003), Pattern based analysis of EAI languages - the case of the business modeling language, *in* O. Camp & M. Piattini, eds, 'Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS 2003)', Vol. 3, Escola Superior de Tecnologia do Instituto Politecnico de Setubal, Angers, France, pp. 174–184.
- Wohed, P., van der Aalst, W., Dumas, M., ter Hofstede, A. & Russell, N. (2005), Pattern-based analysis of UML activity diagrams, *in* 'Proceedings of the 25th International Conference on Conceptual Modeling (ER'2005)', Springer, Klagenfurt, Austria.
- zur Muehlen, M. & Rosemann, M. (2004), Multi-paradigm process management, *in* J. Grundspenkis & M. Kirikova, eds, 'Proceedings of the Fifth Workshop on Business Process Modeling, Development, and Support (BPMDS '04), held in conjunction with the Conference on Advanced Information Systems Engineering (CAiSE) 2004', Vol. 2, Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, pp. 169–175.

Component-Driven Engineering of Database Applications

Klaus-Dieter Schewe¹

Bernhard Thalheim²

¹Massey University, Department of Information Systems & Information Science Research Centre
Private Bag 11 222, Palmerston North, New Zealand, email: k.d.schewe@massey.ac.nz

²Christian Albrechts University Kiel, Department of Computer Science and Applied Mathematics
Olshausenstr. 40, 24098 Kiel, Germany, email: thalheim@is.informatik.uni-kiel.de

Abstract

Though it is commonly agreed that the design of large database schemata requires group effort, database design from component subschemata has not been investigated thoroughly. In this paper we investigate snowflake-like subschemata of database schemata expressed in the Higher-order Entity-Relationship Model (HERM). These subschemata are almost hierarchical in the sense that they may contain cycles in the schema, but not in the instances. We show that each HERM schema can be decomposed into such subschemata using a small set of composition constructors. We then describe how the composition of components can be seen as a database design primitive leading to component-driven database design and re-design pragmatics.

1 Introduction

While design and manufacturing from components is standard in civil, electrical and mechanical engineering, it is still in an embryonal state in software engineering (Arsanjani 2002). In general, component-based engineering means the decomposition of a task, the isolated realisation of the tasks each resulting in a component of the complete system, the composition or “assembly” of the components based on standardised principles.

In program design the design from components has made some progress (Barroca, Hall & Hall 2000, Crnkovic, Hnich, Jonson & Kiziltan 2002) based on clear input/output interfaces. A similar approach has been followed in the emerging area of web services (Hamadi & Benatallah 2003). For database applications, however, design from component subschemata has not been investigated thoroughly. The few existing approaches such as (Akoka & Comyn-Wattiau 1994, Bancillon & Spyrtos 1981, Hay 1995, Jaeschke, Oberweis & Stucky 1994, Rauh & Stickel 1992, Teorey, Wei, Bolton & Koenig 1989) concentrate mainly on the integration of schemata, whereas according to (Thalheim 2000a) the design of database application has to consider also interfaces and dynamic behaviour. Thus, the problem we face in component-based engineering of database applications is deeper, as we have to take care of complex structures, constraints, views and operations.

In this paper we develop an approach to this problem extending and formalising previous work in (Thalheim 2002, Thalheim 2005). We start with a

rather informal discussion in Section 2 on the rationale behind the desire to develop database applications from components. In particular, we discuss problems arising with large schemata, schema patterns observed in practical applications, and the spectrum of different understandings of the term “component”.

Then we investigate components in the higher-order Entity-Relationship model (HERM) (Thalheim 2000a), because the ER-approach is widely used in practice and easy to use, while HERM does not share the deficiencies of some ER-variants such as lack of formal foundations, constraint theory, retrieval and update languages, etc. We present a brief overview of HERM in Section 3 as much as this is necessary for our purposes. We are confident that our approach can be generalised to data models with cyclic references, e.g. sophisticated object models (Schewe & Thalheim 1993) or XML (Abiteboul, Buneman & Suciu 2000).

From various application projects we observe that HERM schemata tend to have larger parts that have the form of star and snowflake schemata, i.e. rather simple schemata centered around a central database type. Such schemata are well known from the area of data warehousing and on-line analytical processing (OLAP) systems. In particular, these subschemata are (almost) hierarchical and correspond to certain tasks within the application. Therefore, we take such subschemata in a generalised form as the basis for components. In particular, we do not request that cycles are completely absent, but that cycles may occur in the schema, but not in the instance, which can be expressed by simple path constraints. This is similar to γ -acyclicity in databases (Hegner 1988). Furthermore, we extend these subschemata with the necessary “plugs” that are used to amalgamate them in a way that behaviour defined for a component carries over to behaviour on the amalgam. We develop the formal basics of this theory of components in Section 4. In particular, the “plugs” will be formalised by (updatable) views and operations on these views.

On this basis we develop a composition theory in Section 5, which basically consists of a collection of composition operations. These generalise input/output behaviour for program modules. We then show that each HERM schemata is in fact the composition of its maximal snowflake components.

This decomposition theorem is central, as it shows that design from snowflake components can always be achieved. However, we need not only such a theoretical statement, but also pragmatic guidelines for component-driven design, which will consist of pragmatics of setting up (not necessarily maximal) snowflake components as in (Feyer & Thalheim 2002), the process of amalgamation, and the assessment of the resulting interface (Vestnický, Lewerenz & Feyer 2000). In particular, we prefer components with

Copyright ©2006, Australian Computer Society, Inc. This paper appeared at the Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), University of Tasmania, Hobart, Australia. Conferences in Research and Practice in Information Technology, Vol. 53. Markus Stumptner, Sven Hartmann, and Yasushi Kiyoki, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

minimal overlap. This pragmatic approach to the design will be addressed in Section 6.

2 Rationale for Component-Driven Application Development

Large database schemata can be drastically simplified if techniques of modular modeling such as *design by units* (Thalheim 2000a) are used. Modular modeling is an abstraction technique based on principles of hiding and encapsulation. Design by units allows to consider parts of the schema in a separate fashion. The parts are connected via types which function similar to bridges.

Data warehousing and user views are often based on snowflake or star schemata. The intuition behind such schemata is often hidden.

Codesign (Thalheim 2000a) of database applications aims in consistent development of all facets of database applications: structuring of the database by schema types and static integrity constraints, behavior modeling by specification of functionality and dynamic integrity constraints and interactivity modeling by assigning views to activities of actors in the corresponding dialogue steps. Codesign, thus, is based on the specification of the the database schema, functions, views and dialogue steps. At the same time, various abstraction layers are separated such as the conceptual layer, requirements acquisition layer and implementation layer.

Codesign is a rather complex procedure. If, however, a component-based approach is used it becomes rather simpler. First, a skeleton of components is developed. This skeleton can be refined during evolution of the schema. Then, each component is developed step by step. If this component is associated to another component then its development must be associated with the development of the other component as long as their common elements are concerned.

Therefore, structuring in codesign may be based on two constructs:

Components: Components are the main building blocks. They are used for structuring of the main data. The association among components is based on ‘connector’ types (called hinge or bridge types) that enable in associating the components in a variable fashion.

Skeleton-based construction: Components are assembled together by application of connector types. These connector types are usually relationship types.

The term *component* has been around for a long time. Component-based software has become a “buzzword” since about ten years beyond classical programming paradigms such as structured programming, user-defined data types, functional programming, object-orientation, logic programming, active objects and agents, distributed systems and concurrency, and middleware and coordination. Various component technologies have been developed since then:

- Source-level language extensions: CORBA, JavaBeans;
- Binary-level object models: OLE, COM, COM+, DCOM, .NET;
- Compound documents: OLE, OpenDoc, Black-Box;
- All-promising design tools: UML, Rational Rose.

A component is considered to be a software implementation that can be autonomously executed, implements one or more interfaces, has a system-wide identity, has instances with their own identity, bundles data and procedures and hides the details of the implementation that are irrelevant to the outside.

The components usually used are considered to be small programs. In reality, a component is a basic unit which can be separated. Therefore, the size might be larger than usually considered in COM+ programming.

Object-orientation has led to a better culture in software projects. It has led to a number of conceptions that are widely used in database applications such as object identifier, rich type systems, active objects, triggers, and polymorphism. At the same time limitations of these concepts have been investigated, e.g., pitfalls of identifiability (Beeri & Thalheim 1999) or rule triggering (Schewe & Thalheim 1998).

Object-orientation has led to a large number of pitfalls (Webster 1995) which substantially reduced its usefulness. Object-orientation is not well-suited for component-based development and hinders it (Nierstrasz & Meijler 1995). Object-oriented source code exposes the class hierarchy and not the interaction among objects. Therefore objects are wired instead of plugged together. The association of objects is distributed among the objects. Object-orientation is domain-driven and leads to designs based on domain objects instead of available components and standard architectures. Rich object interfaces are used instead of plug-compatible interfaces. Furthermore, compositional abstractions such as synchronization policies, coordination abstractions, wrappers and mixins (Ancona & Zucca 1998) cannot be naturally handled as objects.

2.1 Problems with Large Schemata

Monographs and database course books usually base explanations on small or ‘toy’ examples. Reality is, however, completely different. Database schemata tend to be large, unsurveyable, incomprehensible and partially inconsistent due to application, the database development life cycle and due to the number of team members involved at different time intervals. Thus, consistent management of the database schema might become a nightmare and may lead to legacy problems. The size of the schemata may be very large. Comparing and surveying the database schemata brought to our knowledge we observed a high similarity within the solutions. Thus, we have systematized the schemata in (Thalheim 2000b). There is a considerable effort for handling large schemata. In (Moody 2001) most of the methods proposed so far have been generalized to *map abstractions*. Large schemata must be represented in a way that improves understanding and supports documentation and maintenance. Psychological studies have shown that limitations of the short-term memory result in limited capacities for processing and surveying information. The seven-plus/minus-two paradigm applies to database modeling as well. (Maier 1996) reported that enterprise data models consist in the mean of 536 entity types. (Moody 2001) observed therefore that the size of such models may be the reason for the poor understanding of ER modeling in practice. He discovered further that diagrams quickly become unreadable once the number of entity and relationship types exceeds about twenty.

It is a common observation that large database schemata are error-prone, difficult to maintain and to extend and not-surveyable. Moreover, development of retrieval and operation facilities requires highest professional skills in abstraction, memorization and

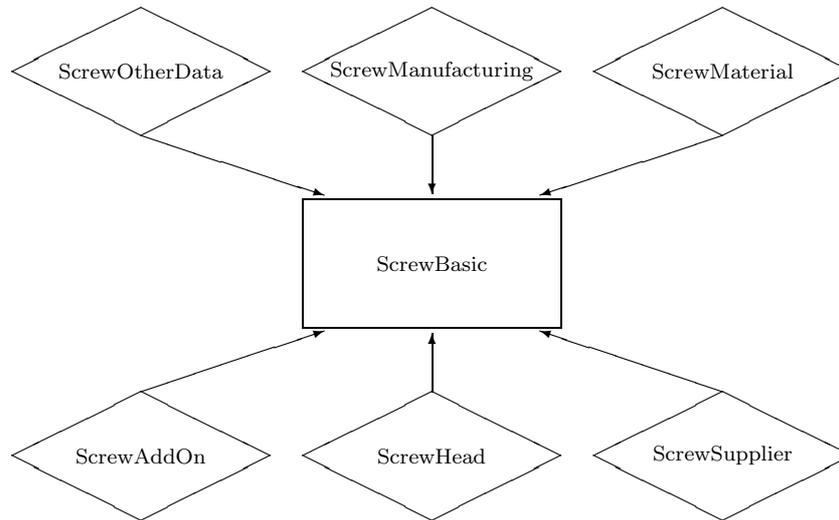


Figure 1: HERM Representation of the Star Type Screw

programming. Such schemata reach sizes of more than 1000 attribute, entity and relationship types. Since they are not comprehensible any change to the schema is performed by extending the schema and thus making it even more complex. Database designers and programmers are not able to capture the schema. Systems such as SAP R/3 are highly repetitive and redundant. For this reason, performance decreases due to bad schema design.

Application schemata could be simpler only to a certain extent if software engineering approaches are applied. The repetition and redundancy in schemata is also caused by

- different usage of similar types of the schema,
- minor and small differences of the types structure in application views and
- semantic differences of variants of types.

Therefore, we need approaches which allow to reason on repeating structures inside schemata, on semantic differences and differences in usage of objects.

Large schemata also suffer from the *deficiency of variation detection*: The same or similar content is often repeated in a schema without noticing it. The Lufthansa cargo database schema contains, for instance, several sub-schemata which store very similar information on the transport log and accounting: air transport of goods, ground transport through cooperating companies and ground transport on the airports. These three kinds of transport have been modeled and implemented by three differently located teams. The similarity of the schemata has not been detected by the teams and caused a number of redundancy and inconsistency problems.

2.2 Observations Made In Practice

Psychologists claim that humans are able to perceive 5 ± 2 concepts at the same time. Humans are better in recognizing connected or associated concepts. Thus, it seems that schemata similar to a star are easier to perceive and to understand.

Star typing has been used already for a long time outside the database community. Let us consider the example in Figure 1. It shows a part of the standardized description of screws using in mechanical engineering. Each screw is characterized by basic data. Additionally, properties on the manufacturer, suppliers, material, form such as head etc. may be added.

Thus, a *star type* is characterized by a kernel entity type used for storing basic data, by a number of subtypes of the entity type which are used for additional properties. These additional properties are clustered according to their occurrence for the things under consideration.

This observation has been taken into account by the OLAP community. (Kimball 1996) claims that ER modeling is completely wrong and that database modeling should be based on star and snowflakes in the sense OLAP people are using it. This claim is far too strict. A typical star schema is displayed in Figure 2. This star schema can be obtained via the following view definition (Thalheim 2000a).

In the same fashion the *snowflake schema*, displayed partially without attributes in Figure 3, can be generated on a schema used for representing the information structure on purchases.

It has been shown in (Lewerenz, Schewe & Thalheim 1999) that instead of that views on ER schemata are used. The main problem of the OLAP approach is the high imposed redundancy due to missing basic data and due to extensive maintenance of views. The observations, however, leading to such claims are made by practitioners. They observed that often star or snowflake schemata are used in practical database operating. Combining their observation and (Lewerenz et al. 1999), we find that star and snowflake schemata are often required and can be supported by database views in traditional way.

If the number of views imposed by such requirements is becoming too high then the conceptual schema may cause maintenance problems. Instead of that we propose to “sternify” the conceptual schema. We usually have several alternatives for representation of an application by conceptual schemata. In this paper we develop a theory of star and snowflake sub-schemata within the conceptual schema which allows easy and computationally simple construction of the corresponding view.

Star structuring and snowflake structuring is becoming popular in the XML community. Modeling with optional parts is a typical approach used for XML structures. The cohesion of elements must be expressed through constraints. Star structuring allows to express cohesion of elements by developing a subtype that contains the coexisting elements.

Main hierarchies considered in database modeling are the specialization and generalization hierarchies which are usually coupled by each other. Other exam-

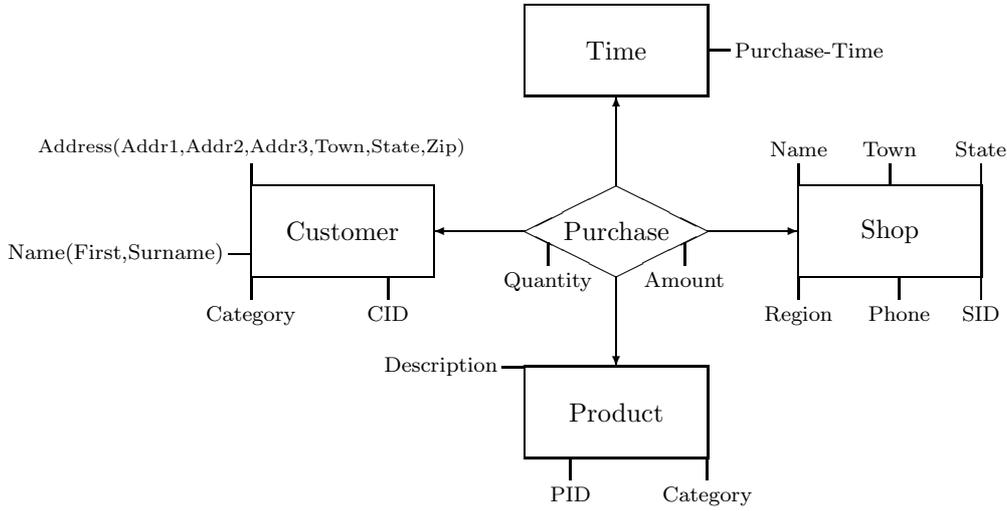


Figure 2: Star Schema on Purchases

ples of hierarchies are the taxonomy, the part-of and the is-part-of hierarchy. Ontologies are often based on hierarchies. Classifications are typical examples of orthogonal hierarchies.

The hierarchical ER model (Silverston, Inmon & Graziano 1997) allows to draw subtypes of entity types as rectangles inside the entity type. A similar representation is star representation. The star sub-schema representation has the advantage that hierarchies of arbitrary depth can be drawn.

3 Enhanced Database Schemata

The major extensions of HERM compared with the flat ER model concern nested attribute structures, higher-order relationship types and clusters, a sophisticated language of integrity constraints, operations, dialogues and their embedding in development methods. In this section we describe HERM (Thalheim 2000a) in a nutshell.

3.1 Static Schemata

In the following let \mathcal{A} denote some set of *simple attributes*. Each simple attribute $A \in \mathcal{A}$ is associated with a base type $dom(A)$, which is some fixed countable set of values. The values themselves are of no particular interest.

In HERM it is permitted to define nested attributes. For this take a set \mathcal{L} of *labels* with the only restriction that labels must be different from simple attributes, i.e. $\mathcal{L} \cap \mathcal{A} = \emptyset$. A *nested attribute* is either a simple attribute, the null attribute \perp , a tuple attribute $X(A_1, \dots, A_n)$ with pairwise different nested attributes A_i and a label $X \in \mathcal{L}$, a list attribute $X[A]$ with a nested attribute A and a label $X \in \mathcal{L}$ or a set attribute $X\{A\}$ with $A \in \mathcal{N}\mathcal{A}$ and $X \in \mathcal{L}$. Let $\mathcal{N}\mathcal{A}$ denote the set of all nested attributes.

We extend *dom* to nested attributes in the standard way, i.e. a tuple attribute will be associated with a tuple type, a set attribute with a set type, and the null attribute with $dom(\perp) = \mathbb{1}$, where $\mathbb{1}$ is the trivial domain with only one value. That is, HERM uses a type system $t = b \mid t_1 \times \dots \times t_n \mid \{t\} \mid [t]$ consisting of base types, record types, set types and list types.

On nested attributes we have a partial order \geq defined as the smallest partial order on $\mathcal{N}\mathcal{A}$ with $A \geq \perp$ for all $A \in \mathcal{N}\mathcal{A}$, $X\{A\} \geq X\{A'\} \Leftrightarrow A \geq A'$, $X[A] \geq X[A'] \Leftrightarrow A \geq A'$, and $X(A_1, \dots, A_n) \geq X(A'_1, \dots, A'_m) \Leftrightarrow \bigwedge_{1 \leq i \leq m} A_i \geq A'_i$.

A *generalized subset* of a set $F \subseteq \mathcal{N}\mathcal{A}$ of nested attributes is a set $G \subseteq \mathcal{N}\mathcal{A}$ of nested attributes such that for each $A' \in G$ there is some $A \in F$ with $A \geq A'$. It is easy to see that $X \geq X'$ gives rise to a canonical projection $\pi_{X'}^X : dom(X) \rightarrow dom(X')$.

Let us now define the entity and relationship types and clusters in HERM. A *level- k -type* R consists of a set $comp(R) = \{r_1 : R_1, \dots, r_n : R_n\}$ of labelled components with pairwise different labels r_i , a set $attr(R) = \{A_1, \dots, A_m\}$ of nested attributes and a key $key(R)$. Each component R_i is a type (or cluster) of a level at most $k-1$, but at least one of the R_i must have level $k-1$. For the key we have $key(R) = comp'(R) \cup attr'(R)$ with $comp'(R) \subseteq comp(R)$ and a generalized subset $attr'(R)$ of the set of attributes. A *level- k -cluster* has the form $R = (r_1 : R_1) \oplus \dots \oplus (r_n : R_n)$ with pairwise different labels r_i and database types R_i of a level at most k , but at least one of the R_i must have level k .

The labels r_i used in components are called *roles*. Roles can be omitted in case the components are pairwise different. A level-0-type E – here the definition implies $comp(E) = \emptyset$ – is usually called an *entity type*, a level- k -type R with $k > 0$ is called a *relationship type*.

A *HERM schema* is a finite set \mathcal{S} of database types and clusters together with a set Σ of integrity constraints defined on \mathcal{S} . We write (\mathcal{S}, Σ) for a schema, or simply \mathcal{S} , if $\Sigma = \emptyset$. We look at constraints in the next subsection.

In order to define the semantics of HERM schemata we associate with each type $R \in \mathcal{S}$ a *representing nested attribute* $\bar{X}_R = R(X_{R_1}, \dots, X_{R_n}, A_1, \dots, A_m)$ as well as a *key attribute* $K_R = R(K_{R_{i_1}}, \dots, K_{R_{i_\ell}}, A'_1, \dots, A'_m)$ for $key(R) = \{r_{i_1} : R_{i_1}, \dots, r_{i_\ell} : R_{i_\ell}, A'_1, \dots, A'_m\}$. With a cluster we associate $\bar{X}_R = (r_1 : \bar{X}_{R_1}) \oplus \dots \oplus (r_n : \bar{X}_{R_n})$ and $K_R = (r_1 : K_{R_1}) \oplus \dots \oplus (r_n : K_{R_n})$, using an additional union-type constructor \oplus . Obviously, we have $\bar{X}_R \geq K_R$ in all cases.

An *instance* of a HERM schema (\mathcal{S}, Σ) is a family $\{db(R)\}_{R \in \mathcal{S}}$ of finite sets $db(R)$ with elements of type \bar{X}_R subject to the obvious key and referential integrity constraints and the explicit constraints in Σ . Note that a whole instance is itself a value of type $DB\{(d_1 : R_1) \oplus \dots \oplus (d_k : R_k)\}$ (for $\mathcal{S} = \{R_1, \dots, R_k\}$). According to this, we may use a rather relaxed notion of *subschema*: a HERM schema that defines a subattribute of the attribute associate with the whole schema, i.e. we can omit types, or attributes or replace attributes by subattributes.

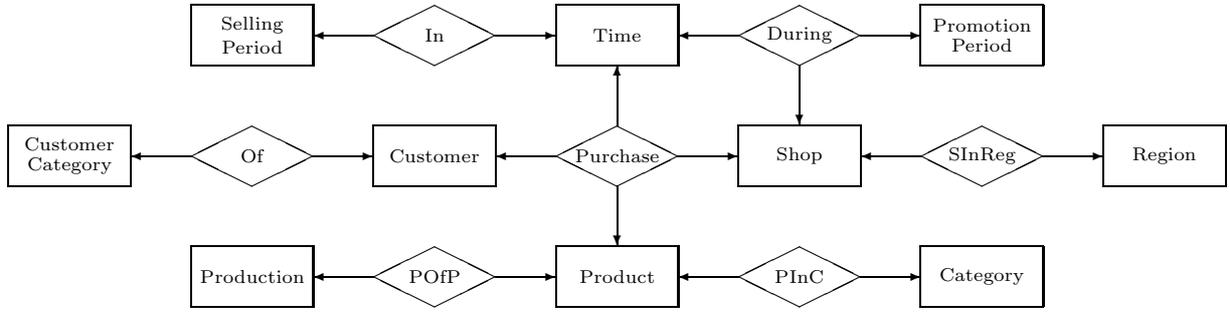


Figure 3: Snowflake Schema on Purchases

3.2 Constraints

Constraints on a HERM schema \mathcal{S} are expressed as formulae in a typed higher-order logic. The types are those defined by the type system above including the union types. Then for each type t we assume a set V_t of variables. In particular, we can use the names of the database types and clusters in the schema as variables: R is considered a variable of type $\{dom(X_R)\}$.

Using such variables we may define terms and formulae as follows:

- Each variable of type t is also a term of type t .
- If τ is a term of type t and $t \geq t'$ holds, then $\pi_{t'}^t(\tau)$ is a term of type t' .
- If τ_1, \dots, τ_n are terms of type t_1, \dots, t_n , respectively, then (τ_1, \dots, τ_n) is a term of type $t_1 \times \dots \times t_n$.
- If φ is a formula and x a variable of type t , then $\mathbf{I}x.\varphi$ is a term of type t .
- If τ is a term of type $\{t\}$ or $[t]$ and ϱ is a term of type t , then $\varrho \in \tau$ is a formula.
- If τ and ϱ are terms of the same type t , then $\varrho = \tau$ is a formula.
- If φ and ψ are formulae and x is a variable, then $\varphi \wedge \psi$, $\neg\varphi$ and $\exists x.\varphi$ are formulae.

The interpretation of these formulae is standard. We should only remark that $\mathbf{I}x.\varphi$ denotes the unique value for x satisfying φ . This gives a powerful instrument to introduce a lot more terms and formulae as shortcuts.

If \mathcal{S} is a HERM schema, then a *constraint* on \mathcal{S} is a formula φ with free variables only among \mathcal{S} . Hence, a constraint on \mathcal{S} can be evaluated in an instance of \mathcal{S} , which we already exploited in the previous subsection.

3.3 Behaviour

In order to add dynamics to HERM schemata, we define operations. In general, an operation is defined by an input schema I , an output schema O and a body. The body works on a working schema W with both I and O as subschemata. It defines a transformation of instances of I into instances of W using an expression of the extended HERM algebra. An operation that preserves the instance of I , i.e. does not change it, is a *query*.

The HERM algebra uses operations for all types of the type system including structural recursion on sets and lists (Tannen, Buneman & Wong 1992) plus a generalised join-operation. In (Schewe 2001) it was shown that this covers all complex value algebras that have been defined so far. The extension of the HERM algebra consists of assignments $X := exp$, where X

is a variable and exp an expression of the same type t , such that exp can be evaluated on instances of W , sequencing of such assignments, and iteration **while change do** (sequence).

If the input schema I consists of a single database type R , we obtain operations on R . Adding operations on all $R \in \mathcal{S}$ and operations with input schema \mathcal{S} we obtain a dynamic HERM schema. However, our major interest is on extended views.

A *view* V on a HERM schema \mathcal{S} is defined by a query q_V , i.e. by an operation with input schema \mathcal{S} , output schema \mathcal{S}_V and an operation body that preserves instances of \mathcal{S} . We extend each such view V with a set \mathcal{O}_V of operations defined on the input schema \mathcal{S}_V . We write (V, \mathcal{O}_V) to denote such an extended view. A *behavioural schema* for a HERM schema \mathcal{S} consists a set of extended views (V, \mathcal{O}_V) on \mathcal{S} .

Operations in \mathcal{O}_V may preserve the input, i.e. the instance of \mathcal{S}_V . Such operations are *retrieval* operations. The other operations are *update* operations. An update operation o_V must translate into an operation o on the database schema \mathcal{S} , i.e. there must exist an operation on \mathcal{S} with $q_V \circ o = o_V \circ q_V$. This operation o must be *generic* in the sense that it commutes with all database automorphisms.

A view (V, \mathcal{O}_V) with at least one update operation in \mathcal{O}_V is called an *input view*. A view with only retrieval operations in \mathcal{O}_V is called an *output view*.

The motivation behind this terminology is the following. The operations on a view define the (local) behaviour of the database. If we want to construct a database schema plus a behavioural schema by means of components, we must be able to integrate not only the structures, which is easy by requesting that the overlap defines a view, but also the behavioural schemata. That is, if a user executes an update-operation on some view V , this will lead to a change of the database instance for the schema \mathcal{S}_V . If this view is also a view on another database schema, the update affects the other database schema. Thus, we can only expect to integrate the behaviour in a consistent way, if an output view on one schema is plugged into an input view on the other one. In other words, the views function in the same way as communication channels. We will exploit this idea in our definition of components in the next section.

4 Almost Hierarchical Components

In this section we develop the basics of our theory of components starting from (generalised) snowflake schemata. We will then define schema components formally, and characterise the compatibility of components, i.e. when two components can be integrated.

4.1 Snowflake Schemata

If \mathcal{S} is a HERM schema, we say that $R_1, R_2 \in \mathcal{S}$ are *connected* iff R_1 is a component of R_2 or vice versa. In this way we obtain a symmetric, irreflexive binary relation \sim on \mathcal{S} . A subschema \mathcal{S}' of \mathcal{S} is a *strictly hierarchical subschema* or *snowflake schema* with center type $R \in \mathcal{S}'$ iff it forms a rooted tree with respect to \sim and root R .

If the depth of the tree is one, we obtain the special case of a *star schema*. For these we distinguish *specialisation schemata*, in which the center type is a component of the other types, *generalization schemata*, in which the center type is a cluster and the other types are components of it, and *association schemata*, in which case the center type is a relationship type and the other types are among its components.

Strictly hierarchical subschemata are not common. However, most schemata in practice have larger subschemata that are almost hierarchical, i.e. types other than the center type may still be connected, but these connections are governed by constraints that guarantee that the induced cycle does not cause harm. We say that a subschema \mathcal{S}' is *almost hierarchical* iff it has a center type $R_0 \in \mathcal{S}'$ such that for all paths p, p' from R_0 to any other type $R_k \in \mathcal{S}'$ one of the following conditions holds:

- Either p and p' lead to entirely different entries of the database, i.e. the exclusion dependency $p[R_0, R_k] \parallel p'[R_0, R_k]$ is implied by the constraints Σ on \mathcal{S} ;
- or p and p' are completely identical, i.e. the pairing inclusion constraints $p[R_0, R_k] \subseteq p'[R_0, R_k]$ and $p[R_0, R_k] \supseteq p'[R_0, R_k]$ are implied by the constraints Σ on \mathcal{S} .

The exclusion constraints allow us to rename the non-identical types to obtain the tree required for snowflake schemata, because in this case, the paths carry different meanings. Similarly, the pairing inclusion constraints allow us to cut the last association in the second path and to obtain an equivalent schema this way or to introduce a mirror type R'_k for the second path in order to obtain the tree required for snowflake schemata. In this case, the paths carry identical meaning.

However, we do not have to execute these schema manipulation operations. It is sufficient to know that it is in principle possible to transform an almost hierarchical subschema into a strictly hierarchical subschema without loss of semantics. Therefore, from now on we call almost hierarchical subschemata also *snowflake schemata*.

4.2 Components

Using the idea from the end of the previous section we can now formally define the notion of component. A *component* \mathcal{C} of a HERM schema (\mathcal{S}, Σ) is a subschema (\mathcal{S}', Σ') together with a set \mathcal{I} of input-views (V_I, \mathcal{O}_{V_I}) and a set \mathcal{D} of output-views (V_O, \mathcal{O}_{V_O}) . We write $\mathcal{C} = (\mathcal{S}', \Sigma', \mathcal{I}, \mathcal{D})$. If (\mathcal{S}', Σ') is a snowflake subschema, we call the component \mathcal{C} a *snowflake component*.

We now have to formalise our idea of channel indicated in the previous section in order to obtain a condition for the compatibility of components. For this assume two components $\mathcal{C}_i = (\mathcal{S}'_i, \Sigma'_i, \mathcal{I}_i, \mathcal{D}_i)$ ($i = 1, 2$). If we integrate the structures (and constraints) of both components, we obtain a new subschema (\mathcal{S}', Σ') such that both \mathcal{S}'_i become views over

\mathcal{S}' via defining queries q_i , i.e. we have induced transformation $q_i : inst(\mathcal{S}', \Sigma') \rightarrow inst(\mathcal{S}'_i, \Sigma'_i)$ ($i = 1, 2$).

Now let us consider a common view (V, \mathcal{O}_V) , i.e. we have induced transformations $q_{V_i} : inst(\mathcal{S}'_i, \Sigma'_i) \rightarrow inst(S_V)$. Hence $q_{V_i} \circ q_i$ induces a defining query on (\mathcal{S}', Σ') for this view.

Consider an operation $o_V \in \mathcal{O}_V$. If it is a retrieval operation, there is no problem. However, if it is an update operation for \mathcal{S}'_1 , we have a transformation $o_V : inst(S_V) \rightarrow inst(S_V)$ that translates into a database transformation $o_1 : inst(\mathcal{S}'_1, \Sigma'_1) \rightarrow inst(\mathcal{S}'_1, \Sigma'_1)$ such that $q_{V_1} \circ o_1 = o_V \circ q_{V_1}$ holds. If o_V is only a retrieval operation for \mathcal{S}'_2 , the operation o_1 is already the translation of o_V for the integrated schema (\mathcal{S}', Σ') . If o_V is also an update operation for \mathcal{S}'_2 , we obtain a translation $o_2 : inst(\mathcal{S}'_2, \Sigma'_2) \rightarrow inst(\mathcal{S}'_2, \Sigma'_2)$ with $q_{V_2} \circ o_2 = o_V \circ q_{V_2}$ in the same way. Then $o_1 \oplus o_2 : inst(\mathcal{S}', \Sigma') \rightarrow inst(\mathcal{S}', \Sigma')$ defines the translation on the integrated schema.

Operations on views define the local behaviour for each component. As we have seen this translates into behaviour on integrated components, if we can either identify views or if views of one component have nothing in common with views on the other component. This outlines a necessary condition that has to be satisfied by composition operations.

5 Composition of Components

So far we defined the notion of component focusing on structure, constraints and behaviour. We showed that we have to be able to identify views on components that are needed for defining the behaviour. This gives the guideline for defining composition operations as the backbone of a composition theory, which we will outline in this section.

5.1 Composition Operations

In the following we define unary and binary operations on components. For this let $\mathcal{C} = (\mathcal{S}', \Sigma', \mathcal{I}, \mathcal{D})$ and $\mathcal{C}_i = (\mathcal{S}'_i, \Sigma'_i, \mathcal{I}_i, \mathcal{D}_i)$ ($i = 1, 2$) be components.

Unification of channels: The component $\zeta_{V_2:=V_1}(\mathcal{C})$ is obtained from \mathcal{C} by unifying the views V_1 and V_2 and using only the names of V_1 . In order to be applicable this operation requires compatible views in terms of the view schemata and operations. Operations in \mathcal{O}_{V_1} are preserved, whereas operation in \mathcal{O}_{V_2} may be added to the new view (after renaming).

Permutation of channels: The component $\tau_{V_1, V_2}(\mathcal{C})$ is obtained from \mathcal{C} by interchanging the views V_1 and V_2 . In order to be applicable this operation requires compatible views in terms of the view schemata and operations.

Renaming of channels: The component $\delta_{\mathfrak{Y}, \mathfrak{Y}'}(\mathcal{C})$ is obtained from \mathcal{C} by replacing the views in \mathfrak{Y} by those in \mathfrak{Y}' . In order to be applicable this operation requires a mapping $\rho : \mathfrak{Y} \rightarrow \mathfrak{Y}'$ that maps each view in \mathfrak{Y} to a compatible view in \mathfrak{Y}' in terms of the view schemata and operations. Furthermore, the new views in \mathfrak{Y}' must not have name conflicts with \mathcal{C} .

Introduction of additional channels: The component $\iota_{\mathfrak{Y}}(\mathcal{C})$ is obtained from \mathcal{C} by adding views in \mathfrak{Y} to those in \mathcal{I} and \mathcal{D} . In order to be applicable this operation requires the new views in \mathfrak{Y} must not have name conflicts with \mathcal{C} nor any interference with the existing views.

Parallel composition with feedback: The component $\mathcal{C}_1 \otimes \mathcal{C}_2 = (\mathcal{S}', \Sigma', \mathcal{I}, \mathcal{D})$ is obtained from \mathcal{C}_1 and \mathcal{C}_2 in the following way:

- The schema (\mathcal{S}', Σ') is the union of the schemata $(\mathcal{S}'_1, \Sigma'_1)$ and $(\mathcal{S}'_2, \Sigma'_2)$.
- The input views are $\mathcal{I} = (\mathcal{I}_1 \cup \mathcal{I}_2) - (\mathcal{D}_1 \cup \mathcal{D}_2)$.
- The output views are $\mathcal{D} = (\mathcal{D}_1 \cup \mathcal{D}_2) - (\mathcal{I}_1 \cup \mathcal{I}_2)$.
- The views in $\mathcal{Z} = (\mathcal{I}_1 \cap \mathcal{D}_2) \cup (\mathcal{D}_1 \cap \mathcal{I}_2)$ define internal view cooperation.

In order to be applicable this operation requires that the two initial components do not have name conflicts, i.e. views are either common to both schemata or completely isolated. Note that we do not request the views to be completely integrated. Instead of this we let the views that couple the behaviour of the two components cooperate with each other.

According to (Thalheim 2000a) view cooperation – as used in the definition of parallel composition with feedback – is a perfect alternative to view integration. Equivalently, we may transform the schema of $\mathcal{C}_1 \otimes \mathcal{C}_2$ into an integrated schema thereby dispensing with the internal view cooperation.

5.2 Amalgams

In the previous subsection we described how to compose components to obtain larger integrated components. We left it open, whether we really integrate components or prefer them to cooperate. The latter one emphasises the independence of local components, whereas the former one emphasise the global aspects. This choice is reflected in the notion of an amalgam.

An *amalgam* \mathcal{A} is defined by a set $\{\mathcal{C}_1, \dots, \mathcal{C}_n\}$ of components, a composition prescription, i.e. an algebraic expression $amg(\mathcal{C}_1, \dots, \mathcal{C}_n)$ built from the composition operation above and the components \mathcal{C}_i ($i = 1, \dots, n$), and a set \mathcal{V} of views defined on $amg(\mathcal{C}_1, \dots, \mathcal{C}_n)$.

The set \mathcal{V} contains the input and output views resulting from the composition of $\mathcal{C}_1, \dots, \mathcal{C}_n$ according to the composition prescription. We classify these views into two sets \mathcal{V}_{int} and \mathcal{V}_{ext} . \mathcal{V}_{int} contains views that are internal to the amalgam, i.e. they will never be used in a composition with further components. That is, these views isolate the local behaviour of the component $amg(\mathcal{C}_1, \dots, \mathcal{C}_n)$. \mathcal{V}_{ext} contains views that are external to the amalgam, i.e. they can be identified with views on other components, thus reflect global behaviour. We call \mathcal{V}_{ext} the *hide* of the amalgam \mathcal{A} and denote it by $hd(\mathcal{A})$.

Speaking pragmatically, amalgams are what we expect to obtain from designing a well-defined part of a database application system. So, if a development project is decomposed into tasks, each task should result in an amalgam, and the composition of all the amalgams gives the complete system.

Therefore, if we are given an amalgam \mathcal{A} , then the integrated schema (\mathcal{S}, Σ) resulting from the composition expression together with the hide $hd(\mathcal{A})$ define a behaviour-extended HERM schema, which we call the *amalgamation* of \mathcal{A} .

5.3 Example

The approach has been applied in the *Cottbus-Interactive* project that delivers video on demand, tv/radio on demand, internet on profile, electronic

program guides on profile. The customers use TV extended by interactive set-top-boxes that provide cable channel services, interaction facilities, and internet services. The later are based on web content that is collected on the basis of the usage statistics of *Cottbus-Interactive* users. The platform is financed through billing customers of some forms of web content and through generation of anonymous web usage profiles. So, we developed amalgams for management of person information, for management of billing, for acquisition of usage records of customers, for utilization of web content, and for elicitation of marketing statistics.

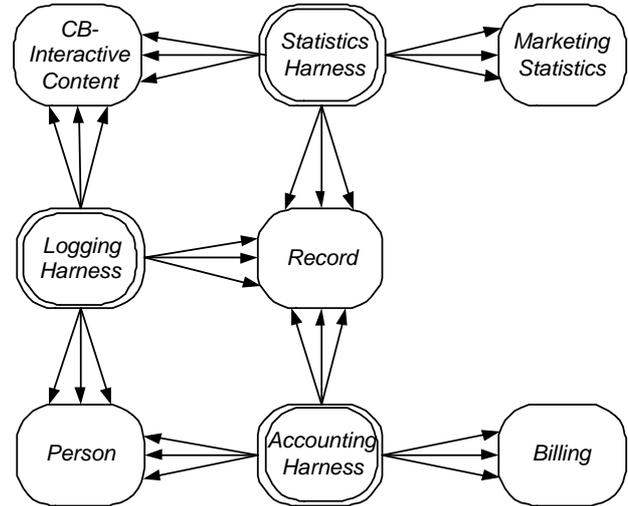


Figure 4: The Architecture of the *Cottbus-Interactive* Project

The amalgams use a number of interfaces as illustrated in Figure 4. The *Person* amalgam provides two interfaces for abstract identification of customers and correspondingly for identification of customers, accounting and posting. The *Record* amalgam has two aggregation interfaces, one for providing summarized utilization records and another one for usage categorization. These four interfaces are read-only interfaces. Additionally the last amalgam provides a read-write interface for collection of records such as logs. The *Web content* amalgam provides an identification interface and a categorization interface. The *Marketing statistics* amalgam has beside a number of read-only interfaces a read-write interface that allows to insert new statistics.

5.4 Decomposition Theorem

We started our motivation with the observation that larger parts of HERM schemata almost look like snowflake schemata. Now that we formalised what “almost” means in this case, we formally want to verify this observation. For this we take a HERM schema (\mathcal{S}, Σ) extended by a behavioural schema, i.e. a set \mathcal{V} of extended views (V, \mathcal{O}_V) on \mathcal{S} . We write $(\mathcal{S}, \Sigma, \mathcal{V})$ for this behaviour-extended schema.

THEOREM 5.1 *Each behaviour-extended HERM schema $(\mathcal{S}, \Sigma, \mathcal{V})$ is the amalgamation of its maximal snowflake components.*

PROOF (SKETCH): Take all the maximal snowflake subschemata $(\mathcal{S}_1, \Sigma_1), \dots, (\mathcal{S}_k, \Sigma_k)$ of (\mathcal{S}, Σ) . Then for each $i = 1, \dots, k$ consider the views in $(V, \mathcal{O}_V) \in \mathcal{V}$ that are defined on $(\mathcal{S}_i, \Sigma_i)$, i.e. the defining query q_V only requires input from this subschema. This

defines sets of views $\mathcal{V}_1, \dots, \mathcal{V}_k$. By separating these views into input and output views we obtain maximal snowflake components $\mathcal{C}_i = (\mathcal{S}_i, \Sigma_i, \mathcal{I}_i, \mathcal{O}_i)$.

As (\mathcal{S}, Σ) is the integration of the subschemata $(\mathcal{S}_i, \Sigma_i)$, it can be represented by a composition expression $amg(\mathcal{C}_1, \dots, \mathcal{C}_k)$. So we get the desired composition prescription, and $\mathcal{A} = (\{\mathcal{C}_1, \dots, \mathcal{C}_k\}, amg(\mathcal{C}_1, \dots, \mathcal{C}_k), \mathcal{V})$ is the desired amalgam. The hide $hd(\mathcal{A})$ of this amalgam is the set of views in \mathcal{V} that do not belong to one of the view sets $\mathcal{V}_1, \dots, \mathcal{V}_k$. \square

6 Component-Based Design

The decomposition theorem in the previous section shows that component-driven design is always possible. This gives a theoretical underpinning for methodological considerations. However, for practical purposes the theorem is of minor value. It guarantees the decomposition into maximal snowflake components, whereas pragmatically we would prefer to have components with minimal overlap – which implies maximal autonomy – and bound to tasks in the application. Therefore, we develop now a pragmatic approach to component-driven design consisting of four phases:

1. We start from behaviour-extended schemata for certain tasks of the application, as they may arise from cutting up a development project and then working independently. These schemata may not be snowflake components. However, by the decomposition theorem they can be represented as amalgams. Then the definition of views that connect these components defines an amalgam for the whole application.
2. Each component resulting from step one can be decomposed into snowflake components by the decomposition theorem. So we need algorithms for detecting components and checking, whether they are almost hierarchical or not. Together with phase one this amounts to an amalgam with a larger number of components, but these components are now snowflakes.
3. In the third phase we consider the overlap between components aiming at minimising them as much as possible. The result will still be an amalgam with snowflake components, but these components do not overlap excessively any more.
4. Finally, we reconsider the components resulting from phase three and recombine some of them, if this the result is still a snowflake component and the application considers the initial components as belonging together to one task of the application.

In the sequel we develop these phases in more detail. Of course the connection to application tasks can only be decided in an application context. However, we can look for the technical issues involved in these phases. These are the detection of components, the transformation of behaviour-extended subschemata into amalgams, and the minimisation of overlap between components.

We start with a HERM schema (\mathcal{S}, Σ) , from which we first derive a graph \mathcal{G} with \mathcal{S} as its vertex set and the edges defined by the connection relation \sim . However, if $R' \in \mathcal{S}$ occurs more than once as a component of $R \in \mathcal{S}$, we get multiple edges in \mathcal{G} . Using well-known algorithms we choose a spanning tree for this graph \mathcal{G} . This defines a colouring of the edges in \mathcal{G} : green edges occur in the spanning tree and red edges are the remaining ones.

Next consider path exclusion and path pairing inclusion dependencies derived from Σ . Let Σ_p be the set of these dependencies. For each cycle in \mathcal{G} that is composed out of two paths, one containing only green, the other one containing only red edges, check whether it is subject to a constraint in Σ_p . If this is not the case, the two vertices connected by the “red” path cannot belong to a snowflake subschema. In this way, we partition \mathcal{S} into subschemata $\mathcal{S}_1, \dots, \mathcal{S}_m$ such that each \mathcal{S}_i defines a connected subgraph of \mathcal{G} , in which all vertices are pairwise connected by a “green” path. Let Σ_i be the projection of Σ onto the subschema \mathcal{S}_i . Then the subschemata $(\mathcal{S}_i, \Sigma_i)$ ($i = 1, \dots, m$) are snowflake subschemata.

Of course, this approach to discovering snowflake subschemata depends on the choice of the spanning tree. If a chosen spanning tree does not produce a satisfactory set of snowflake subschemata, we may have to repeat the procedure with a different spanning. This increases complexity, but we gain a more suitable decomposition.

Furthermore, it is not necessary to choose maximal subschemata $(\mathcal{S}_i, \Sigma_i)$. We may still decide to decompose some of the \mathcal{S}_i with or without overlap. This, however, is again a pragmatic decision.

Once we have discovered snowflake subschemata, we have to turn them into components to make up the component set of an amalgam. This basically consists of taking those views V of the global behavioural schema with a defining query defined on the subschema \mathcal{S}_i to $(\mathcal{S}_i, \Sigma_i)$. The classification into input and output views then defines components $\mathcal{C}_i = (\mathcal{S}_i, \Sigma_i, \mathcal{I}_i, \mathcal{O}_i)$. All the remaining views become part of the amalgam hide.

The remaining step is to discover the composition prescription that will turn $\mathcal{C}_1, \dots, \mathcal{C}_m$ into \mathcal{S} . As we decomposed \mathcal{S} using a spanning tree in the associated graph \mathcal{G} , this amounts to rearranging operations that are not defined on one component \mathcal{C}_i .

Finally, for the minimisation of component overlap we can assume that the HERM schema (\mathcal{S}, Σ) is the amalgamation of an amalgam \mathcal{A} with components $\mathcal{C}_i = (\mathcal{S}_i, \Sigma_i, \mathcal{I}_i, \mathcal{O}_i)$ ($i = 1, \dots, m$). Consider two of these components, say \mathcal{C}_1 and \mathcal{C}_2 without loss of generality. We may adopt the approach in (Klettke 1999).

7 Component-Based Abstraction

Proponents of the information framework, e.g., (Evernden 1994), have already claimed that conceptual models should be considered to be low-level or tactical models. At the enterprise level, a high level view of data needs to be established that *strategically* guides the development and deployment of the information system. The ‘ballpark view’ concentrates on an architectural understanding of the application. Component-based design may be considered to be the first perspective solution to strategic information frameworks. It introduces a new physical architecture or a new IS architecture paradigm and easily adopts new paradigms such as web services.

Component-based design uses the idea of high level schema conceptions as a way of thinking about the application. Business users do not have to look beyond to the underlying detailed schema but rather focus understanding on the types of things that make up the application solution. *Component-based abstraction* is based on the previously discussed component model, amalgamation, high-level presentation of architectures, and mappings to conceptual schemata.

The solution displayed in Figure 4 shows that component-based development supports reverse engineering and re-engineering of applications. Re-

engineering is considered to be the examination and alteration of an information system to reconstitute it in a new form and the subsequent implementation of the new form. This process encompasses a combination of sub-processes such as reverse engineering, forward engineering, analysis, synthesis, migration, rephrasing etc. Reverse engineering based on component abstraction raises the level of abstraction of a schema through e.g. decompilation, sub-schema extraction, architecture recovery, documentation generation, and visualization. Refactoring based on component-based abstraction improves the IS design by restructuring it such that it becomes easier to understand while preserving its structuring and functionality. Migration transform the schema to another one at the same level of abstraction based on conversion or translation. Rephrasing transforms a database schema into a different one based on normalization and optimization. Component-based abstraction is an invasive solution. The challenge is decomposing the monolithic plain schema of the usually fully structured legacy system to the richly hierarchic and structured component architecture.

The Cottbus-interactive project solution uses five high level conceptions: content, person, record, billing, and statistics. These components can be understood as highly generic business concepts that define the scope of the application being designed. They combine internal hierarchies (Bekke 1992). Hierarchies used are of classification, meta-data, and internal association. For instance, a person may be classified as 'individual' or 'customer', for statistics as 'single' and 'married'. The meta-data hierarchy is used for addition of properties of the kernel objects such as context of use, descriptors, time restrictions, utilization or copyright issues. Internal associations within a component support detailing of relationships among types of a component. The components are associated in a variety of different relationships. We use harnesses for their combination. Similar to receptors and ligands, we use views for associating components.

Component-based abstraction achieves four targets that could not be solved in classical information systems engineering:

IS understanding: High level abstraction based on components provide information for maintenance or redevelopment support teams, e.g., dependencies between components, harnesses, etc.

IS quality assurance: components are developed with higher initial efforts and thus enforce design standards or perform "bad smells" detection for types used.

IS transformation and optimization:

Component-based design supports defect detection and fixing, performance optimizations (e.g. bad type elimination), and design structure improvements. The IS schema is transform to a modern architecture while keeping the core IS structuring and functionality.

Integration of different IS: The integration may be based on information portals, schemata assimilation, replication, based on shared components, on exchange services such as SOA or based on distributed components.

8 Conclusion

In this paper we presented a HERM-based theory of component-driven database application design. The simple guideline of our work is that components are structurally simple, which can be represented by a

slight generalisation of snowflake schemata. Furthermore, components isolate "local" behaviour, which can be expressed by operations on views.

Our composition theory consists of composition operations that take care of the integration of structure, constraints and behaviour. We demonstrated that each schema results from the composition of snowflake components. Reversing this theoretical result we presented a pragmatics-driven approach to design components with minimal overlap.

Our next steps will be to generalise the approach from HERM to other data models, in particular those that permit cyclic references. Furthermore, we aim at extending and refining the pragmatics side of our approach.

References

- Abiteboul, S., Buneman, P. & Suci, D. (2000), *Data on the Web: From Relations to Semistructured Data and XML*, Morgan Kaufmann Publishers.
- Akoka, J. & Comyn-Wattiau, I. (1994), A framework for automatic clustering of semantic models, in Elmasri, Kouramajian & Thalheim (1994), pp. 438–450.
- Ancona, D. & Zucca, E. (1998), 'A theory of mixin modules: Basic and derived operators', *Mathematical Structures in Computer Science* **8**(4), 401–446.
- Arsanjani, A. (2002), 'Developing and integrating enterprise components and services', *CACM* **45**(10), 30–34.
- Bancilhon, F. & Spyrtos, N. (1981), Independent components of databases, in 'Very Large Data Bases, 7th Int. Conf.', IEEE Press, Cannes, France, pp. 398–408.
- Barroca, L., Hall, J. & Hall, P. (2000), *Software architectures - Advances and applications*, Springer, Berlin.
- Beeri, C. & Thalheim, B. (1999), Identification as a primitive of database models, in T. Polle, T. Ripke & K.-D. Schewe, eds, 'Proc. Fundamentals of Information Systems, 7th Int. Workshop on Foundations of Models and Languages for Data and Objects - FoMLaDO'98', Kluwer, London, Timmel, Ostfriesland, pp. 19–36.
- Bekke, J. H. T. (1992), *Semantic data modelling*, Prentice-Hall, London.
- Crnkovic, I., Hnich, B., Jonson, T. & Kiziltan, Z. (2002), 'Specification, implementation and deployment of components', *CACM* **45**(10), 35–40.
- Elmasri, R., Kouramajian, V. & Thalheim, B., eds (1994), *Proc. 12th Int. ER Conf., Entity-Relationship Approach - ER'93*, LNCS 823, Springer, Berlin, Arlington, USA, Dec. 15 - 17, 1993.
- Evernden, R. (1994), 'The information framework', *IBM Systems Journal* **35**(1), 37–68.
- Feyer, T. & Thalheim, B. (2002), Many-dimensional schema modeling, in Y. Manolopoulos & P. Návrát, eds, 'ADBIS', Vol. 2435 of LNCS, Springer, pp. 305–318.
- Hamadi, R. & Benatallah, B. (2003), A petri net-based model for web service composition, in 'ADC', pp. 191–200.

- Hay, D. C. (1995), *Data model pattern: Conventions of thought*, Dorset House, New York.
- Hegner, S. J. (1988), Decomposition of relational schemata into components defined by both projection and restriction, in 'Proc. 7th ACM SIGACT-SIGMOS-SIGART Symp. on Principles of Database Systems - PODS'88', ACM Press, New York, Austin, Texas, pp. 174–183.
- Jaeschke, P., Oberweis, A. & Stucky, W. (1994), Extending ER model clustering by relationship clustering, in Elmasri et al. (1994), pp. 451–462.
- Kimball, R. (1996), *The data warehouse toolkit*, John Wiley & Sons, New York.
- Klettke, M. (1999), Reuse of database design decisions, in P. P. Chen, D. W. Embley, J. Kouloumdjian, S. W. Liddle & J. F. Roddick, eds, 'Advances in Conceptual Modeling – Proceedings of the ER 1999 Workshops', LNCS 1727, Springer, Berlin, pp. 213–224.
- Lewerenz, J., Schewe, K.-D. & Thalheim, B. (1999), Modeling data warehouses and olap applications by means of dialogue objects, LNCS 1728, Springer, Berlin, Paris, France, Nov. 15–18, 1999, pp. 354–368.
- Maier, R. (1996), Benefits and quality of data modeling - results of an empirical analysis, LNCS 1157, Springer, Berlin, Cottbus, Germany, Oct. 7 - 10, 1996, pp. 245–260.
- Moody, D. L. (2001), Dealing with complexity: A practical method for representing large entity-relationship models, PhD thesis, Dept. of Information Systems, University of Melbourne.
- Nierstrasz, O. & Meijler, T. D. (1995), 'Research directions in software composition', *ACM Computing Surveys* **27**(2), 262–264.
- Rauh, O. & Stickel, E. (1992), Entity tree clustering - a method for simplifying er designs, LNCS 645, Springer, Berlin, Karlsruhe, Germany, Oct. 7 - 9, 1992, pp. 62–78.
- Schewe, K.-D. (2001), On the unification of query algebras and their extension to rational tree structures, in M. Orłowska & J. Roddick, eds, 'Proc. Australasian Database Conference 2001', Australian Computer Society, pp. 52–59.
- Schewe, K.-D. & Thalheim, B. (1993), 'Fundamental concepts of object oriented databases', *Acta Cybernetica* **11**(4), 49–84.
- Schewe, K.-D. & Thalheim, B. (1998), 'Limitations of rule triggering systems for integrity maintenance in the context of transition specification', *Acta Cybernetica* **13**, 277–304.
- Silverston, L., Inmon, W. H. & Graziano, K. (1997), *The data model resource book*, John Wiley & Sons, New York.
- Tannen, V., Buneman, P. & Wong, L. (1992), Naturally embedded query languages, in 'ICDT', pp. 140–154.
- Teorey, T. J., Wei, G., Bolton, D. L. & Koenig, J. A. (1989), 'ER model clustering as an aid for user communication and documentation in database design', *CACM* **32**(8), 975–987.
- Thalheim, B. (2000a), *Entity-Relationship Modeling: Foundations of Database Technology*, Springer-Verlag.
- Thalheim, B. (2000b), The person, organization, product, production, ordering, delivery, invoice, accounting, budgeting and human resources pattern in database design, Technical Report Preprint I-07-2000, Brandenburg University of Technology at Cottbus, Institute of Computer Science. See also: <http://www.is.informatik.uni-kiel.de/~thalheim/slides.htm>.
- Thalheim, B. (2002), Component construction of database schemes, in 'ER', pp. 20–34.
- Thalheim, B. (2005), 'Component development and construction for database design', *Data Knowl. Eng.* **54**(1), 77–95.
- Vestenicky, V., Lewerenz, J. & Feyer, T. (2000), Modeling the modification component of an information service, in 'Proc. of Challenges, ADBIS-DASFAA 2000, Prague', pp. 195–204.
- Webster, B. F. (1995), *Pitfalls of object-oriented development: a guide for the wary and enthusiastic*, M&T books, New York.

Supporting Virtual Organisation Alliances with Relative Workflows

Xiaohui Zhao, Chengfei Liu and Yun Yang

Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, VIC 3122, Australia

{xzhaoh, cliu, yyang}@it.swin.edu.au

Abstract

Driven by the fast changing service demand-and-supply requirements, virtual organisation alliances are created to adapt highly dynamic B2B collaborations. However, the temporary partnership and low trustiness between collaborating organisations raise challenges to effectively manage collaborative business processes. This paper presents an approach on the basis of a service oriented relative workflow model to support virtual organisation alliances. This approach takes an organisation centred design method and deploys a visibility mechanism to provide a finer granularity of authority control at contracting and collaboration design phases. The open contracting and inter-organisational collaboration design in a virtual organisation alliance are particularly addressed and discussed in this paper.

Keywords: business process modelling, service oriented computing, virtual organisation alliance.

1 Introduction

Recent years witnessed the trend of global business collaboration which urgently requires organisations to dynamically form virtual organisation alliances. A virtual organisation alliance seamlessly integrates the business processes of different organisations to adapt the continuously changing business conditions and to stay competitive in the global market (Osterle, Fleisch and Alt 2001, van der Aalst and van Hee 2004, zur Muehlen 2004).

Different from the large-scale organisations centred virtual enterprises, a virtual organisation alliance is mainly constructed from small-to-medium sized organisations, which join a virtual community to share each other's business services. The collaborations in such a virtual organisation alliance are always motivated by prompt business service demand-and-supply requirements, such as service outsourcing or business service complementation. The collaborations are rather temporary and dynamic ones. Therefore, a pre-fixed inter-organisational workflow process design mechanism may work very awkwardly in such scenarios. In a virtual organisation alliance, each member organisation needs to

publish and update its business services that can be provided or outsourced. Then, other member organisations can choose partner organisations and create corresponding collaborations to best fit its requirements or profit benefits.

Two characteristics, i.e. the dynamic structure and the collaboration openness, distinguish the virtual organisation alliance from traditional federated organisations. And these two characteristics also raise challenges to manage the collaborative business processes for virtual organisation alliances, especially at contracting and collaboration design phases. The temporary and dynamic cooperation relationship requires high flexibility in describing and implementing collaboration processes between member organisations. Furthermore, the dynamic and temporary partnership in turn results in the lack of trustiness between member organisations in loose-coupling business collaborations, and therefore complicates the authority control (Schulz and Orlowska 2004, Quirchmayr et al. 2002).

Many approaches (Grefen et al. 2001, Kajko-Mattsson 2003, Berfield et al. 2002) attempt to precisely architect a virtual organisation alliance with diagrams at process level, resource level, function level, organisation level, and so forth. But these complex models fail in the flexibility and adaptability towards the characteristics of dynamics and openness. Some approaches implicitly assume or explicitly model (Besembel, Hennet and Chacon 2002) business development functions in the virtual organisation alliances, which are often referred to as "broker", "business architect", "integrator", "project manager" (Katzy and Lon 2003) or similar names. These approaches always adopt an absolute view of collaborations, which presents the same picture of the structure and relationships to every member organisation in a virtual organisation alliance, and therefore neglect the aspects of authority control and privacy respect.

Aiming to solve these problems, this paper extends our previously proposed relative workflow model into the service oriented computing environment to well support the collaboration behaviours of dynamic virtual organisation alliances. This model treats each participating organisation as an autonomous entity, and empowers the organisation to design inter-organisational workflow processes from its own perspective. With regard to the authority control, this organisation oriented mechanism enables the visibility differentiation for different partner organisations in the open collaborating environment of a virtual organisation alliance. In the proposed approach, contracts are not only used to define and regulate business service collaborations, but also to assist developing the visibility constraints for the business process integration.

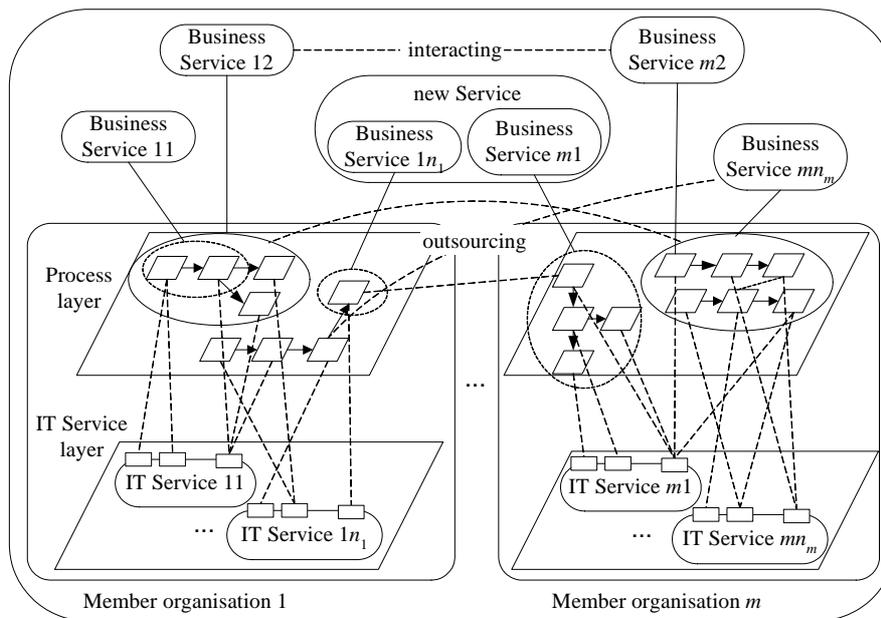


Figure 1: Virtual organisation alliance architecture

The rest of this paper is organised as follows: Section 2 first presents the relation between business services and IT services, and then briefly reviews the proposed relative workflow model with an extension towards service oriented computing. Section 3 discusses how to support business collaborations in the environment of a virtual organisation alliance with the relative workflow model, especially at the phases of contracting and collaboration design. In section 4, an application example is used to demonstrate how to practically apply the relative workflow approach to accommodate dynamic collaborations in a virtual organisation alliance. Conclusion remarks are given in Section 5.

2 Service Oriented Relative Workflow Model

2.1 Business Services and IT Services

Basically, B2B collaborations are motivated by service demand-and-supply requirements, at a high level. And the implementation of collaborations is supported by some fine-grained IT services at a low level. A virtual organisation alliance is established to quickly capture emerging market opportunities and enact business service collaborations at a high level, by the means of utilising and coordinating the functions provided by low-level IT services.

Technically, business services stand for the business related procedures or work that can benefit others. In most cases, such business services are coarse-grained. For example, products manufacturing service, after-sales service etc. of a manufacturing company can be viewed as its business services.

The notion of IT services comes from Service Oriented Computing (SOC), which is emerging as a new computing diagram for distributed computing and business integration. An IT service denotes an Internet-accessible service (Papazoglou and Georgakopoulos 2003, Leymann, Roller and Schmidt 2002). As building blocks of modern

enterprise application architecture, IT services provide a good support on interoperability and flexibility. In this field, some leading companies and organisations, such as IBM, Microsoft and OASIS, have contributed a lot in defining specifications and developing architectures for service oriented computing. Now, Web services are widely considered to be the most popular IT service technology, which uses Web Service Description Language - WSDL (W3C 2001) and Business Process Execution Language for Web Services - BPEL4WS (Andrews et al. 2003) to describe the service interfaces and interaction routines, respectively.

Figure 1 illustrates the architecture of a dynamical virtual organisation alliance, where the collaborations between member organisations are represented in forms of business service interactions, business service outsourcing and business service composition at the top layer. The business services are supported by a single or multiple business processes, which streamline related handling procedures and regulate the usage of involved resources and staff at the intermediate layer, i.e. the process layer. At the bottom layer, i.e. the IT service layer, IT services are invoked through specific operations by the workflow processes at the process layer. The workflow processes streamline the related workflow tasks, and embed the orchestration and choreography of IT service invocations to fulfil a particular business goal. In the service oriented computing architecture shown in Figure 1, workflows and IT services together build up the fundamental infrastructure to support high-level business services.

2.2 Extension of Relative Workflow Model

Basically, traditional workflow modelling approaches assume that there exists a third-party designer or a leading organisation of the collaborating organisations can see certain level of details of all participating organisations. Unfortunately, the predominant view of a third-party designer or a leading organisation seriously violates the privacy protection of participating organisations in a

loosely-coupled virtual organisation alliance. As such, the visibility between participating organisations may be relative, rather than absolute as adopted in the public view approach. Besides, the pre-fixed business collaboration in the public view approach can hardly meet the continuously evolving partnerships between member organisations.

To solve these problems, we proposed the relative workflow model (Zhao, Liu and Yang 2005) to define and enact inter-organisational workflows from a *relative* view rather than an *absolute* view. In this section, we extend the relative workflow model into the service oriented computing environment, by adding two definitions, *IT service* and *business service*. To hide private information during business collaborations, a participating *organisation* is allowed to wrap its *local workflow processes* into a series of *perceivable workflow processes* for different partner organisations, according to the *visibility constraints* defined in corresponding *perceptions*. And a *relative workflow process* is generated by linking an organisation's local workflow processes with perceivable workflow processes of its partner organisations. Different from traditional inter-organisational workflow solutions, relative workflows restrict an organisation's interaction within the scope of its local workflow processes and perceivable workflow processes of partner organisations, so as to prevent excessive information disclosures. By creating a relative workflow process for each partner organisation, a macro business collaboration process can be distributed into several interactions among neighbouring organisations, where each participating organisation acts as an autonomous entity with the total control of its local workflow processes. In this model, IT services work as building blocks to provide the basic supporting functions. And the orchestration and choreography of IT services is embedded in workflow processes.

Some key definitions are given below.

Definition 1 (IT Service) An IT service is a discrete unit of application logic that exposes message-based interfaces suitable for being accessed across a network. An IT service s is defined as a set of operations $\{op_1, op_2, \dots, op_n\}$. Each operation represents a message-based interface of an IT service. The message used by an operation op can be represented as a message description, $m \times \{in, out\}$, where m denotes the name of the message.

Definition 2 (Business Service) A business service bs of an organisation g represents a unit used for business collaborations. A business service is supported by a proper workflow process, which utilises necessary IT services to fulfil a particular business goal. This supporting workflow process may be a composite process, which consists of multiple collaborating local workflow processes.

Definition 3 (Local Workflow Process) A local workflow process lp is defined as a directed acyclic graph $(\mathcal{T}, \mathcal{R})$, where \mathcal{T} is the set of nodes representing the set of tasks, and $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{T}$ is the set of arcs representing the

execution sequence. Here, a task $t \in \mathcal{T}$ may invoke one or more operations of IT services.

Definition 4 (Organisation) An organisation g owns a set of local workflow processes $\{lp^1, lp^2, \dots, lp^n\}$ to support a set of business services. An individual local workflow process lp^i of g is denoted as $g.lp^i$.

During the collaboration, the organisation applies visibility control to protect the critical or private business information of some workflow tasks from entirely exposing to external organisations. Table 1 lists the three basic visibility values defined for business interaction and workflow tracking.

Visibility value	Explanation
Invisible	A task is said invisible to an external organisation, if it is hidden from that organisation.
Trackable	A task is said trackable to an external organisation, if that organisation is allowed to trace the execution status of the task.
Contactable	A task is said contactable to an external organisation, if the task is trackable to that organisation and the task is also allowed to send/receive messages to/from that organisation for the purpose of business interaction.

Table 1: Visibility values

Definition 5 (Visibility Constraint) A visibility constraint vc is defined as a tuple (t, v) , where t denotes a workflow task and $v \in \{Invisible, Trackable, Contactable\}$. A set of visibility constraints \mathcal{VC} defined on a workflow process lp is represented as a set $\{vc:(t, v) \mid \forall t (t \in lp.\mathcal{T})\}$.

Definition 6 (Perception) A perception defines the information related to an inter-organisational interaction of a local workflow process. Once a business service oriented contract is assigned, the corresponding perception can be derived from the contract. The details about the derivation will be discussed later. A perception $p_{g_1}^{g_0,lp}$ of an organisation g_0 's local workflow process lp from another organisation g_1 is defined as $(\mathcal{VC}, \mathcal{MD}, f)$, where

- \mathcal{VC} is a set of visibility constraints defined on $g_0.lp$.

- $\mathcal{MD} \subseteq \mathcal{M} \times \{in, out\}$, is a set of the message descriptions that contains the messages and the passing directions. \mathcal{M} is the set of message names used to represent inter-organisational business activities.

- $f: \mathcal{MD} \rightarrow g_0.lp_{g_1}.\mathcal{T}$ is the mapping from \mathcal{MD} to $g_0.lp_{g_1}.\mathcal{T}$, and $g_0.lp_{g_1}$ is the *perceivable workflow process* of $g_0.lp$ from g_1 .

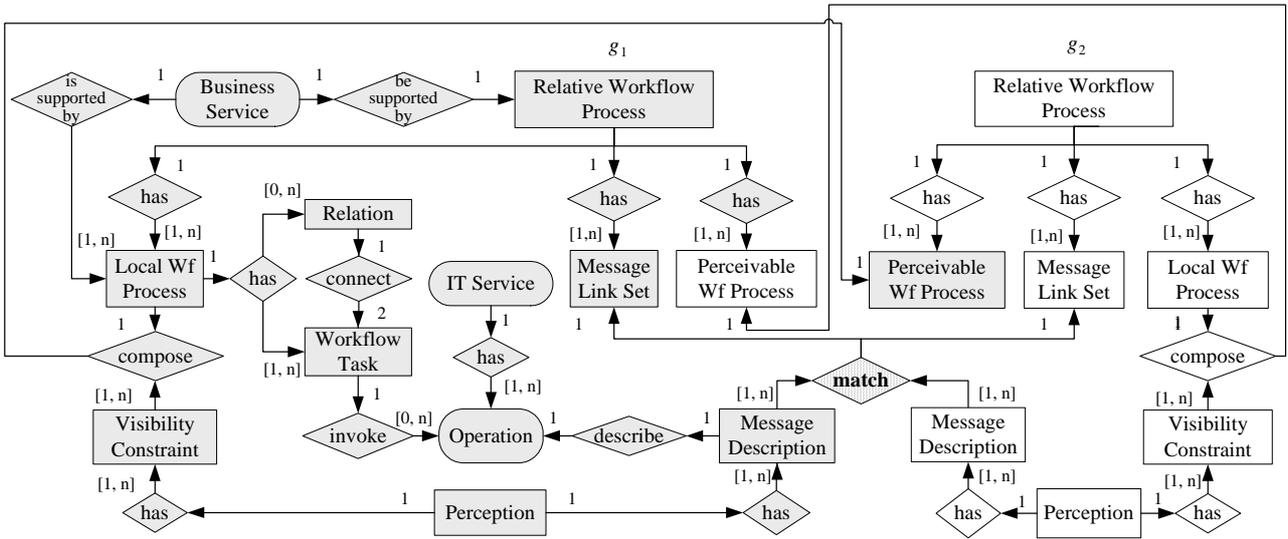


Figure 2: Extended relative workflow model

Definition 7 (Relative Workflow Process) A relative workflow process $g_1.rp$ perceivable from an organisation g_1 is defined as a directed acyclic graph $(\mathcal{T}, \mathcal{R})$, where

\mathcal{T} is the set of the tasks perceivable from g_1 , which is a union of the following two parts:

- $\cup_k g_1.lp^k.\mathcal{T}$, the union of the task sets of all $g_1.lp^k$.
- $\cup_i \cup_j g_i.lp^j_{g_1}.\mathcal{T}$, the union of the task sets of all perceivable workflow processes of $g_i.lp^j$ from g_1 .

\mathcal{R} is the set of arcs perceivable from g_1 , which is a union of the following three parts:

- $\cup_k g_1.lp^k.\mathcal{R}$, the union of the arc sets of all $g_1.lp^k$.
- $\cup_i \cup_j g_i.lp^j_{g_1}.\mathcal{R}$, the union of the arc sets of all perceivable workflow processes of $g_i.lp^j$ from g_1 .

\mathcal{L} , the set of messaging links between local workflow processes and perceivable workflow processes, defined on

$$\cup_i \cup_j \cup_k (g_1.lp^k.\mathcal{T} \times g_i.lp^j_{g_1}.\mathcal{T} \cup g_i.lp^j_{g_1}.\mathcal{T} \times g_1.lp^k.\mathcal{T}).$$

The relative workflow model extended with business and IT services is shown in Figure 2. Given the definitions and discussion above, an organisation, say g_1 , may first establish a business service by conjoining one or more local workflow processes to coordinate related IT services. Once this business service is involved in the collaboration with another organisation, say g_2 , a perceptions can be generated to regulate the visibility control on each involved local workflow processes. Afterwards, g_1 can wrap its local workflow process into an authority safe perceivable workflow process for g_2 , according to the visibility constraints defined in the perception. Finally, at the site of g_2 , a relative workflow process will be assembled from related local workflow processes and perceivable workflow processes from g_1 . Besides, a business service can also be supported by a pre-existing relative workflow process. Correspondingly, this business service may drive the IT services of multiple organisations to work for itself.

3 Supporting Virtual Organisation Alliances

The business collaboration within a virtual organisation alliance can be represented as four phases, viz. contracting, collaboration design, collaboration execution and collaboration termination. This paper focuses on the first two phases, i.e. how to organise business contracting and design inter-organisational business collaborations with the proposed service oriented relative workflow model in the virtual organisation alliance environment.

3.1 Why Relative Workflows Can Support Virtual Organisation Alliances

3.1.1 Support at Contracting Phase

Normally, B2B collaboration originates from contracting, where two or more parties come to an agreement to cooperate for a common objective, and this agreement is regulated by a legal document of contract (Gimpel et al. 2003). (Griffe et al. 1998) have modelled a contract as four major parts of *Who, What, How* and *Legal clauses*. The *How* part defines the execution details for the obligations: when and which services are to be delivered? What is the deadline? Which clause will apply when a party falls behind its obligation? These details together describe the necessary *business interactions* for the collaboration.

Since a virtual organisation alliance enables the collaborations with a broad range of potential partners, each member organisation is empowered to quickly assemble the resources and expertise to capture emerging opportunities. To keep these options open, the partnerships between organisations are not static, but rather continuously evolve to stay competitive on the market. Correspondingly, this open partnership requires an open contracting mechanism, where an organisation posts the business services that it can offer and it may request to all potential co-operators in the virtual organisation alliance. Thereafter, some organisations with special interests may respond by referring to the business services. Finally, the involved organisations can come to negotiate the details of the contract for the collaboration. We call the organisation

that issues the contract is a *host organisation*, and the responding organisations are *partner organisations*.

Different from the traditional closed contracting process, this open contracting process has following features.

- Low trustiness.

Since the contract may be established between parties with no prior partnerships, high trustiness can hardly be granted. The low trustiness requires authority control to prevent potential privacy disclosure during collaborations. As for this issue, the visibility constraint based visibility control mechanism of our relative workflow model is dedicated to guarantee the finer granularity of workflow visibility between cooperating organisations. With these visibility constraints, participating organisations can intentionally choose which tasks to be hidden or revealed to partner organisations according to the level of trustiness and the necessary interactions for collaborations.

- Uni-directional contracting.

Different from the normal contracting process which has defined concrete parties at the starting time, the open contracting process only involves a single party at the beginning, i.e. the host organisation. The uni-directional contracting process can be well supported by the process of posting business services in the context of the service oriented relative workflow model. Once a business service is prepared by deploying underlying supporting workflow processes, it will be published to all other member organisations to seek potential business collaborations. This service posting process also originates from one organisation, i.e. the host organisation, and propagates to all other organisations.

- Agile collaboration.

Because the involved organisations share a loosely-coupled relationship, the collaboration is dynamic with the low coordination, interdependence, short duration and few transactions. The agile collaboration requires the flexibility of collaboration structure and behaviours. Our relative workflow model supports a kind of “off-the-shelf” collaboration formation scheme, which empowers each organisation itself to choose partner organisations and define relative workflow processes from its own local workflow processes and the perceivable workflow process provided by other organisations. In this scheme, each participating organisation acts as an autonomous entity and it can change its partners or redefine its collaborations dynamically, to grasp the fast changing market opportunities.

3.1.2 Support at Collaboration Design Phase

Once a contract is signed by involved parties, participating organisations may come to the next phase, i.e. collaboration design, where each participating organisation designs and coordinates the business collaborations amongst partner organisations by linking related business processes.

At this stage, each member organisation may participate in multiple collaborations with different groups of partner

organisations at the same time. Furthermore, each participating organisations may choose and combine some collaborations into a comprehensive collaboration according to its own preference and management. Hence, different participating organisations may own different forms of business collaborations at the whole picture level. For this reason, the collaboration should be treated from the individual perspective of each participating organisation rather than a public perspective. Upon this point, our relative workflow model adapts the various views from different organisations by designing and maintaining inter-organisational workflow processes from a relative perspective. In consequence, it can better tolerate complicated partnership among participating organisations inside a virtual organisation alliance.

From the above discussion, we can see that our relative workflow model provides a good support to B2B collaborations at contracting and collaboration design phases in the environment of a virtual organisation alliance. The relative perspective of defining and managing inter-organisational workflows features our approach from conventional ones, and the visibility control mechanism and dynamic definition scheme also enhance the authority control and collaboration flexibility. Consequently, relative workflows can particularly serve inter-organisational business collaborations in an open, loosely-coupled and low-trustiness application environment, such as a virtual organisation alliance.

3.2 How Relative Workflows Support Virtual Organisation Alliances with

In the context of relative workflows, inter-organisational business collaborations are initially composed at business service level. At the phase of contracting, organisations first publish their service demand-and-supply requirements. By means of auctions, bidding or free selections, a partner organisation may be determined by the host organisation. And once the contract is negotiated and signed by all the involved parties, the partnership is then confirmed.

Prior to the collaboration design phase, the involved organisations need to set up corresponding perceptions for the authority control in collaborations. For each workflow process related to the contracted business services, a proper perception will be generated according to the signed contract. These perceptions will be used for generating the corresponding perceivable workflow processes. The derivation from business service oriented contracts to workflow process oriented perceptions shall follow a Minimal Disclosure Principle.

Minimal Disclosure Principle: An organisation should keep the disclosure of process details as minimal as possible, relatively to its partnership with different organisations.

The derivation may involve recognising necessary inter-organisational messages, setting up visibility constraints for workflow tasks etc. As mentioned in Section 3.1, a contract c defines the necessary business interactions to fulfil the collaboration. And these business interactions are supported by the invocations through

operations of IT services belonging to cooperating organisations. Algorithm 1 gives the detailed steps on generating a perception p for a local workflow process lp of the host organisation g_0 in the collaboration with the partner organisation g_1 , according to the business interactions defined in c .

Algorithm 1. Generating perceptions

Input:
 c a contract signed by organisation g_0 and g_1
 g_0 the host organisation
 g_1 the partner organisation
 lp an involved local workflow process of g_0

Output:
 p the generated perception on g_0 's lp from g_1

Step 1 Set all tasks invisible.

$p.\mathcal{VC} = \emptyset; p.\mathcal{MD} = \emptyset; p.f = \emptyset;$
for each task $t \in lp$
 $p.\mathcal{VC} = p.\mathcal{VC} \cup \{(t, \text{invisible})\};$
end for

Step 2 Set contactable tasks.

for each business interaction bi defined in contract c
for each operation op invoked by each task $t \in lp$
if op provides necessary functions for business interaction bi **then**
if $\exists (t, \text{invisible}) \in p.\mathcal{VC}$ **then**
 $p.\mathcal{VC} = p.\mathcal{VC} - \{(t, \text{invisible})\};$
 $p.\mathcal{VC} = p.\mathcal{VC} \cup \{(t, \text{contactable})\};$
end if
 $mdSet = \{\text{the message descriptions to be used by } t \text{ to support } bi \text{ via } op\}$
for each message $md \in mdSet$
 $p.\mathcal{MD} = p.\mathcal{MD} \cup \{md\};$
 $(md \rightarrow t) \rightarrow p.f;$
end for
end if
end for

Step 3 Set trackable tasks.

for each business interaction bi defined in contract c
for each task $t \in lp$
if bi has status dependency with t **then**
if $\exists (t, \text{invisible}) \in p.\mathcal{VC}$ **then**
 $p.\mathcal{VC} = p.\mathcal{VC} - \{(t, \text{invisible})\};$
 $p.\mathcal{VC} = p.\mathcal{VC} \cup \{(t, \text{trackable})\};$
end if
end if
end for

Figure 3 illustrates how the open contracting process goes with the underlying perception generation process inside a member organisation.

As Figure 3 shows, an organisation first posts its service demand-and-supply requirements to its virtual organisation alliance. Then another organisation may respond and come to negotiate about the intended collaboration. When a contract is finalised to confirm the collaboration, the visibility filter will generate perceptions and distribute them to involved local workflow processes. Afterwards, the participating organisations come to the collaboration design phase, where a relative workflow

process will be generated to conduct the business collaboration. This relative workflow process integrates the host organisation's local workflow processes and the perceivable workflow processes of partner organisations. The generation of such a relative workflow process involves two operations, i.e. composing tasks and assembling relative workflow processes.

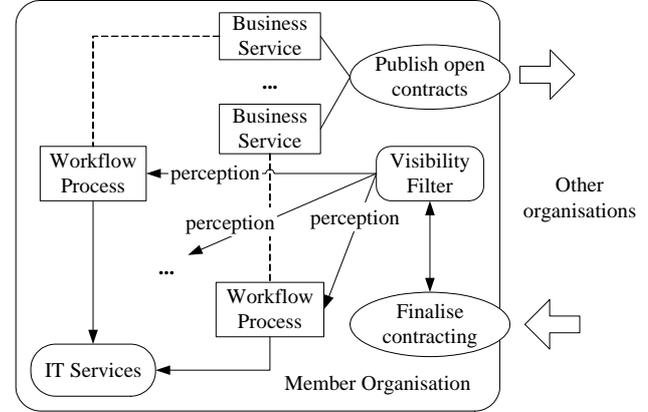


Figure 3: Open relationship contracting

The purpose of composing tasks is to hide private tasks of local workflow processes. We choose to merge invisible tasks with contactable or trackable tasks into composed tasks, if not violating the structural validity; otherwise, those invisible tasks are combined into a dummy task. For example, according to the perception defined from the partner organisation, a local workflow process of the host organisation after this step becomes an authority safe perceivable workflow process. The algorithm for composing tasks is given in Algorithm 2.

For the simplicity of discussion, we only consider composing one local workflow process lp of the organisation g_0 from another organisation g_1 . Furthermore, we conduct a pre-processing on all split/join structures of lp such that for all those branches consisting of only invisible tasks, a dummy task is created to delegate these branches.

Algorithm 2. Composing tasks

Input:
 lp $g_0.lp$, the organisation g_0 's local workflow process lp before composition.
 p $p_{g_1}^{g_0.lp}$, the perception of g_0 's lp from g_1 .

Output:
 lp' $g_0.lp_{g_1}$, the perceivable workflow process composed from lp for g_1 , according to $p_{g_1}^{g_0.lp}$.

Step 1 Connect invisible tasks.

$lp' = lp;$
 $\mathcal{VT} = \{\text{all the visible tasks of } lp, \text{ defined in } p\};$
while $(\exists t, t' \in (lp'.\mathcal{T} - \mathcal{VT})) ((t, t') \in lp'.\mathcal{R}) \wedge \text{seq}(t) \wedge \text{seq}(t')$
// $\text{seq}(t) = (\text{indegree}(t) = 1 \wedge \text{outdegree}(t) = 1)$
{ $t^o = t + t';$
 $lp'.\mathcal{T} = lp'.\mathcal{T} \cup \{t^o\} - \{t, t'\};$
 $lp'.\mathcal{R} = lp'.\mathcal{R} - \{(t, t')\};$
replace t, t' in $lp'.\mathcal{R}$ with t^o ;
}
}

Step 2 Downward composition with incoming interaction

tasks.

```

while (( $\exists t \in \mathcal{V}\mathcal{T}(p'.f^{-1}(t)=(m, in) \wedge outdegree(t)=1) \wedge (\exists t' \in (lp'.\mathcal{T}-\mathcal{V}\mathcal{T}))((t, t') \in lp'.\mathcal{R} \wedge indegree(t')=1)$ ))
{
   $t^\circ = t + t'$ ;
   $\mathcal{V}\mathcal{T} = \mathcal{V}\mathcal{T} \cup \{t^\circ\} - \{t\}$ ;
   $lp'.\mathcal{T} = lp'.\mathcal{T} \cup \{t^\circ\} - \{t, t'\}$ ;
   $lp'.\mathcal{R} = lp'.\mathcal{R} - \{(t, t')\}$ ;
  replace  $t, t'$  in  $lp'.\mathcal{R}$  with  $t^\circ$ ;
}

```

Step 3 Upward composition with outgoing interaction tasks.

```

while (( $\exists t \in \mathcal{V}\mathcal{T}(p'.f^{-1}(t)=(m, out) \wedge indegree(t)=1) \wedge (\exists t' \in (lp'.\mathcal{T}-\mathcal{V}\mathcal{T}))((t, t') \in lp'.\mathcal{R} \wedge outdegree(t')=1)$ ))
{
   $t^\circ = t + t'$ ;
   $\mathcal{V}\mathcal{T} = \mathcal{V}\mathcal{T} \cup \{t^\circ\} - \{t\}$ ;
   $lp'.\mathcal{T} = lp'.\mathcal{T} \cup \{t^\circ\} - \{t, t'\}$ ;
   $lp'.\mathcal{R} = lp'.\mathcal{R} - \{(t, t')\}$ ;
  replace  $t, t'$  in  $lp'.\mathcal{R}$  with  $t^\circ$ ;
}

```

In the operation of assembling relative workflow processes, an organisation may assemble its relative workflow processes from local workflow processes and the perceivable workflow processes of partner organisations, together with the messaging links. The messaging links are obtained by matching the message descriptions defined in perceptions of the host organisation and the partner organisation. Once this relative workflow process is generated, the inter-organisational business service collaboration becomes formally prepared for collaboration execution phase. The corresponding assembling algorithm is given below.

Algorithm 3. Assembling relative workflow processes

Input:

lp' $g_0.lp_{g_1}$, the perceivable workflow process composed from g_0 's local workflow process lp .

P $p_{g_1}^{g_0.lp}$, the perception of g_0 's lp from g_1

ps $\{ p_{g_0}^{g_1.lp^1}, \dots, p_{g_0}^{g_1.lp^{1m}} \}$, the set of perceptions defined on g_1 's perceivable workflow processes from g_0

Output:

\mathcal{L} the set of generated messaging links.

Step Generating messaging links to bind workflow processes.

$\mathcal{L} = \emptyset$;

for each $t \in lp'.\mathcal{T}$

if $\exists md(p.f(md)=t)$ **then** {

$md_1 = p.f^{-1}(t)$;

for each $p^\circ \in ps$

for each $md_2 \in p^\circ.\mathcal{MD}$

if md_1 matches md_2 **then**

$\mathcal{L} = \mathcal{L} \cup \{(t, p^\circ.f(md_2), md_1)\}$;

/ the messaging links are obtained by matching messaging descriptions. */*

}

4 Application Example

Australian toolmaking firms are relatively small and specialised, operating with minimal business infrastructure in an attempt to control overhead costs. This specialisation restricts access to additional customers or larger projects. In response to this increasing dilemma, toolmakers need to become effective in engaging and servicing a more geographically disperse clientele, and complementary toolmakers need to pool their resources. Technology-enabled collaboration can assist with dealing with this industry deficiency. In this section, we attempt to apply our relative workflow approach to support collaboration behaviours of a virtual organisation alliance for these toolmaking firms.

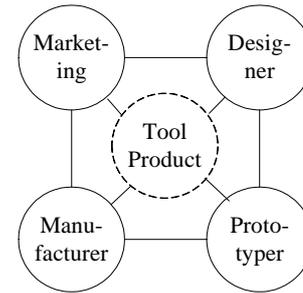


Figure 4: Toolmaking VOA

As Figure 4 shows, a virtual organisation alliance consisting of toolmaking firms may connect designers, manufacturers, prototypers and marketing companies together to collaboratively work for customer products.

With this background, we narrow our focus on a scenario where exist diverse business collaborations between four member organisations, viz. organisation A, B, C and D. Figure 5 illustrates three business collaboration scenarios between the four member organisations. For simplicity, we only give key tasks of the involved workflow processes.

In the scenario of collective production shown in Figure 5, organisation A's production process uses organisation B's production service, which is supported by organisation B's production process. Organisations A and B produce different kinds of parts, respectively, and finally assemble and package them into unitised tools at the site of organisation A. This collaboration is motivated by the production capability requirement, and reflects the synergy for small-to-medium sized organisations.

In the scenario of design outsourcing, organisation A outsources its prototyping task to organisation C, for the efficiency of time and cost, given organisation C provides stronger prototyping services. This collaboration involves the interaction between organisation A's design process and organisation C's prototyping process.

The scenario of bulk ordering shows the economic of scale, since the organisations with orders for the same parts or parts from the same supplier batch their orders together for a more economical price. This collaboration lies between organisation A's ordering process and organisation D's ordering process.

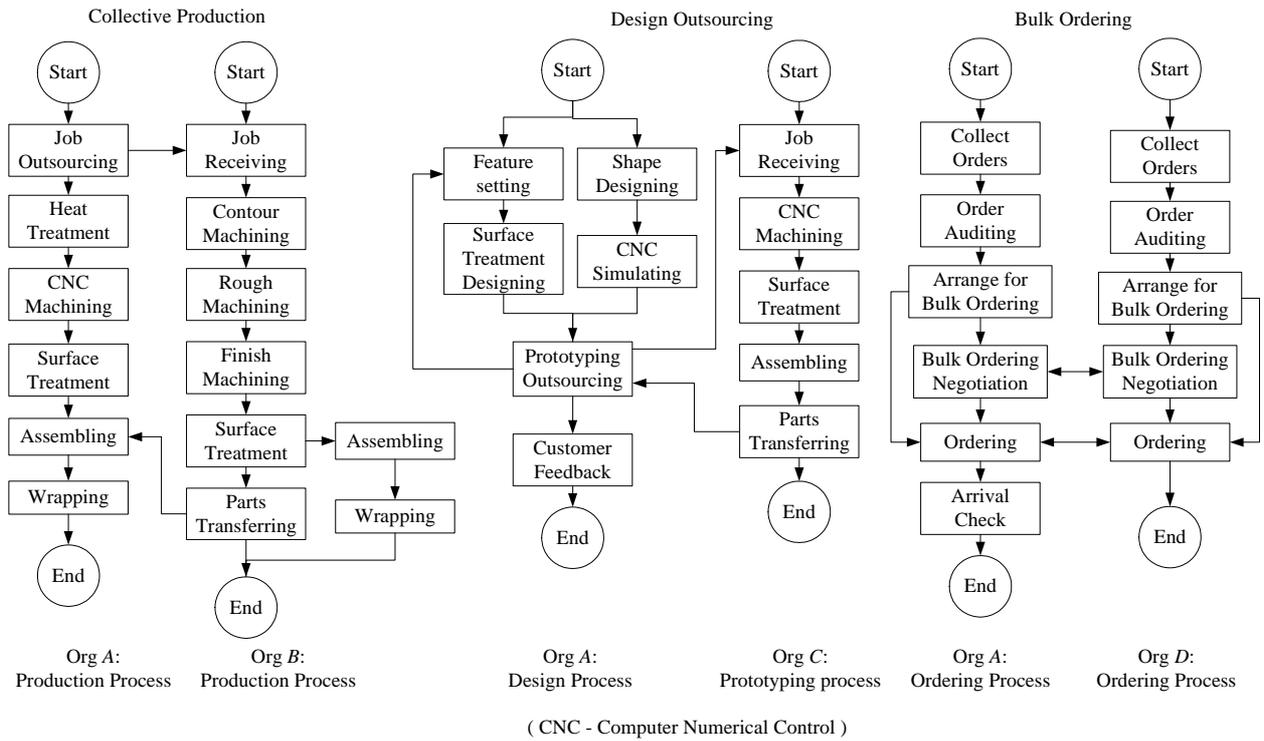


Figure 5: Business collaborations

Now, we start from the bulk ordering collaboration to demonstrate how our relative workflow approach supports a virtual organisation alliance. In the scenario of the bulk ordering collaboration, when organisation A collects orders from its production department(s), it will consider whether to seek a bulk ordering with potential co-buyers. If needed, it will publish a request for bulk ordering of listed parts or materials, to all other member organisations in this alliance. Suppose that organisation D has the same things to buy, and organisation D responds to organisation A to further negotiate the details about the amount for bulk ordering and the expected price, etc. Finally, a contract will be signed to regulate the agreement on bulk ordering, and the two organisations can conjoin their orders. This contract is motivated by seeking an economical price, and the collaboration is supported by the business services of parts ordering of the two organisations, with the underlying supporting workflow processes be organisation A's ordering process and organisation D's ordering process, respectively.

Since this collaboration mainly focuses the bulk ordering negotiation, some tasks of ordering processes may be set invisible for the collaborating organisation, if these tasks do not directly participate in the bulk ordering negotiation. According to the algorithm mentioned in the previous section, the corresponding perception on organisation A's ordering process from the view of organisation D, i.e. $P_D^{A.ordering Process}$, may have the following visibility constraints.

$$\mathcal{V}_1 = \{ ('Collect Orders', Invisible), ('Order Auditing', Invisible), ('Arrange for Bulk Ordering', Invisible), ('Bulk Ordering Negotiation', Contactable), ('Ordering', Contactable), ('Arrival Check', Invisible) \}.$$

These visibility constraints prohibit organisation D's cognition on private tasks, such as "Collect Orders", "Order Auditing" and "Arrange for Bulk Ordering". These tasks only handle internal procedures, and do not participate in the bulk ordering collaboration. Therefore, such prohibition does not affect the negotiation with organisation D.

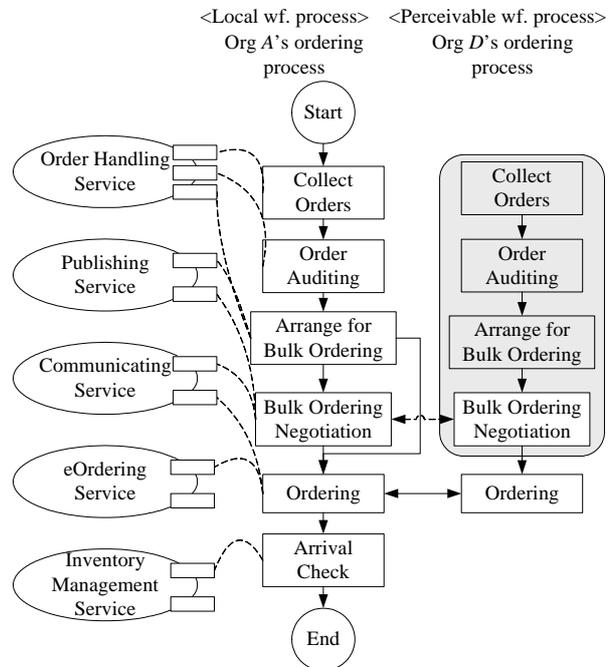


Figure 6: Relative workflow process for bulk ordering collaboration

Similarly, the perception on organisation D's ordering process from the view of organisation A, i.e.

$p_A^{D.ordering Process}$, may have the following visibility constraints.

$\mathcal{V}_2 = \{ ('Collect Orders', Invisible), ('Order Auditing', Invisible), ('Arrange for Bulk Ordering', Invisible), ('Bulk Ordering Negotiation', Contactable), ('Ordering', Contactable) \}$.

Now, we can generate the relative workflow process for this bulk ordering collaboration from the perspective of organisation A, according to the visibility constraints defined in perception $p_A^{D.ordering Process}$. Figure 6 shows the generated relative workflow process.

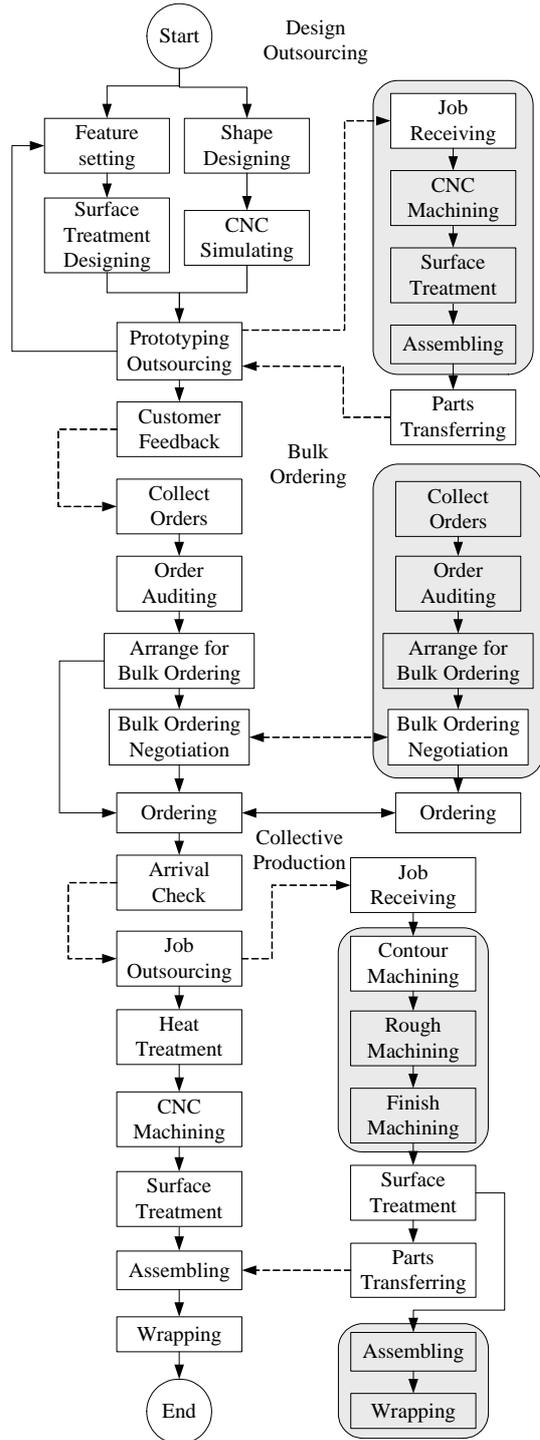


Figure 7: Final relative workflow process

The shadowed tasks of the perceivable workflow process shown in Figure 6 denote the invisible tasks to organisation A, and the white tasks are either trackable or contactable ones. The ovals on the left denote the IT services invoked by organisation A's production process, and the small blank rectangles denote the operations of IT services. Since the two organisations collaborate at process level, and the IT services of organisation D may not be perceivable from organisation A. Therefore, only organisation A's related IT services are given in Figure 6.

Following this way, organisation A may also sign contracts with organisations B and C, for the collective production and design outsourcing. Therefore, organisation A is simultaneously participating in the three collaborations with organisations B, C and D, respectively. And these three collaborations together support organisation A's whole process of tools manufacturing. A composite relative workflow integrating all the three collaborations can be generated at the site of organisation A, to represent organisation A's comprehensive manufacturing business collaboration.

Figure 7 gives the composite relative workflow process from the perspective of organisation A. This relative workflow process combines organisation A's three local workflow processes, i.e. engineering process, ordering process and production process. In addition, this relative workflow process includes three other workflow processes of its partner organisations, i.e. organisation C's prototyping process, organisation D's ordering process and organisation B's production process, in their perceivable forms. For the simplicity, related IT services are not given in Figure 7.

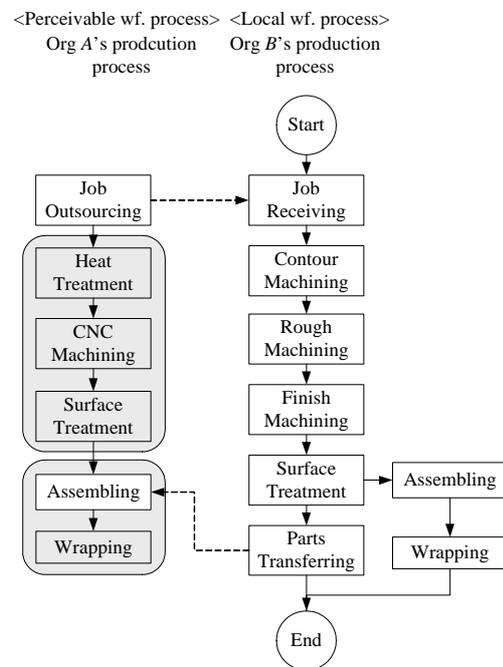


Figure 8: Relative workflow process from org B's view

From the perspective of another participating organisation, say organisation B, it may own a different collaboration picture. Since organisation B does not participate in the collaborations of bulk ordering or design outsourcing with organisation A, organisation B therefore has no authorities

to perceive those two collaborations. This means that organisation *B* may even not know the existence of these two collaborations. The relative workflow generated from the perspective of organisation *B* is given in Figure 8.

From the relative workflow processes shown in Figure 7 and Figure 8, we can see that different organisations hold different views towards collaborations. This reflects our relativity characteristics.

With this proposed approach, each organisation is in charge of choosing partners by issuing and signing proper contracts. In addition, each organisation is responsible for defining the collaboration structure and behaviours to fulfil its own business planning and objective. Each organisation acts as an autonomous entity, and can change its partners or redefine its collaborations dynamically, to grasp the fast changing market opportunities. The visibility control mechanism prevents the private information disclosure at the task level or at the process level. Participating organisations are now able to control the granularity of partner organisations' cognition on its business processes during collaborations.

5 Conclusion

This paper has presented an approach to support B2B collaborations in virtual organisation alliances. With this approach, the inter-organisational collaborations motivated by service demand-and-supply requirements can be interpreted into relative workflow processes that in turn coordinate related IT services at technical level. Different from conventional approaches, our approach adopts a set of visibility control mechanism to restrict the cognition of collaborating organisations on each other's business processes, and therefore is free of privacy disclosure and authority violation. In addition, the organisation oriented design scheme empowers organisations to dynamically choose partner organisations and customise collaboration structures.

6 References

- Andrews, T., Curbera, F., Dholakia, H., Golland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I. and Weerawarana, S. (2003): *Business Process Execution Language for Web Services (BPELWS) 1.1*, <http://www.ibm.com/developerworks/library/ws-bpel>.
- Berfield, A., Chrysanthos, P. K., Tsamardinos, I., Pollack, M. E. and Banerjee, S. (2002): A Scheme for Integrating e-services in Establishing Virtual Enterprises, *Proceedings of Twelfth International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems*, 134-142.
- Besembel, I., Henet, J. C. and Chacon, E. (2002): Coordination by Hierarchical Negotiation within an Enterprise Network, *Proceedings of 8th International Conference on Concurrent Enterprising*, Rome, Italy, 507-516.
- Gimpel, H., Ludwig, H., Dan, A. and Kearney, R. (2003): PANDA: Specifying Policies for Automated Negotiations of Service Contracts, *International Conference on Service-Oriented Computing*, Trento, Italy, 287-302.
- Grefen, P. W. P. J., Aberer, K., Ludwig, H. and Hoffner, Y. (2001): CrossFlow: Cross-Organizational Workflow Management for Service Outsourcing in Dynamic Virtual Enterprises. *IEEE Data Engineering Bulletin*, **24**: 52-57.
- Griffe, F., Boger, M., Weinreich, H., Lamersdorf, W. and Merz, M. (1998): Electronic Contracting with COSMOS - How to Establish, Negotiate and Execute Electronic Contracts on the Internet, *2nd Int. Enterprise Distributed Object Computing Workshop*, 46-55.
- Kajko-Mattsson, M. (2003): Infrastructures of virtual IT enterprises, *Proceedings of International Conference on Software Maintenance*, 199-208.
- Katzy, B. and Lon, H. (2003): Virtual Enterprise Research State of the Art and Ways Forward, *Proceedings of 9th International Conference on Concurrent Enterprising*, Helsinki, Finland.
- Leymann, F., Roller, D. and Schmidt, M. T. (2002): Web Services and Business Process Management. *IBM Systems Journal*, **41**: 198-211.
- Osterle, H., Fleisch, E. and Alt, R. (2001): *Business Networking - Shaping Collaboration between Enterprises*, Springer Verlag.
- Papazoglou, M. P. and Georgakopoulos, D. (2003): Special issue on service oriented computing. *Communications of the ACM*, **10**: 24-28.
- Quirchmayr, G., Milosevic, Z., Tagg, R., Cole, J. B. and Kulkarni, S. (2002): Establishment of Virtual Enterprise Contracts, *Proceedings of 13th International Conference Database and Expert Systems Applications*, 236-248.
- Schulz, K. and Orłowska, M. (2004): Facilitating Cross-organisational Workflows with a Workflow View Approach. *Data & Knowledge Engineering*, **51**: 109-147.
- van der Aalst, W. M. P. and van Hee, K. M. (2004): *Workflow Management: Models, Methods, and Systems*, Cambridge, MA, MIT Press.
- W3C (2001): Web Services Description Language (WSDL) 1.1. <http://www.w3c.org/TR/wsdl>.
- Zhao, X., Liu, C. and Yang, Y. (2005): An Organisational Perspective of Inter-Organisational Workflows, *International Conference on Business Process Management*, 17-31.
- zur Muehlen, M. (2004): *Workflow-based Process Controlling: Foundation, Design, and Application of Workflow-driven Process Information Systems*, Berlin, Logos Verlag.

Author Index

- Benatallah, Boualem, 7
Biddle, Robert, 11
- Colomb, Robert, 75
- de Vries, Denise, 85
- Galloway, John, 21
Governatori, Guido, 75
Gregersen, Heidi, 35
- Hartmann, Sven, iii
- Kitagawa, Takashi, 55
Kiyoki, Yasushi, iii
- Liu, Chengfei, 115
- Malinowski, Elzbieta, 45
Mayr, Heinrich C., 3
Motahari-Nezhad, Hamid Reza, 7
- Nakanishi, Takafumi, 55
Nesbitt, Keith V., 65
Noble, James, 11
- Padmanabhan, Vineet, 75
- Rice, Sally, 85
Roddick, John F., 85
Rotolo, Antonino, 75
Russell, Nick, 95
- Sadiq, Shazia, 75
Schewe, Klaus-Dieter, 105
Simoff, Simeon J., 21
Stumptner, Markus, iii
- ter Hofstede, Arthur H.M., 95
Thalheim, Bernhard, 105
- van der Aalst, Wil M.P., 95
- Wohed, Petia, 95
- Yang, Yun, 115
- Zhao, Xiaohui, 115
Zimányi, Esteban, 45

Recent Volumes in the CRPIT Series

ISSN 1445-1336

Listed below are some of the latest volumes published in the ACS Series *Conferences in Research and Practice in Information Technology*. The full text of most papers (in either PDF or Postscript format) is available at the series website <http://crpit.com>.

- Volume 41 - Theory of Computing 2005**
 Edited by Mike Atkinson, *University of Otago, New Zealand* and Frank Dehne, *Griffith University, Australia*. January, 2005. 1-920-68223-6.
 Contains the papers presented at the Eleventh Computing: The Australasian Theory Symposium (CATS2005), Newcastle, NSW, Australia, January/February 2005.
- Volume 42 - Computing Education 2005**
 Edited by Alison L. Young, *UNITEC, New Zealand* and Denise Tolhurst, *University of New South Wales, Australia*. January, 2005. 1-920-68224-4.
 Contains the papers presented at the Seventh Australasian Computing Education Conference (ACE2005), Newcastle, NSW, Australia, January/February 2005.
- Volume 43 - Conceptual Modelling 2005**
 Edited by Sven Hartmann, *Massey University, New Zealand* and Markus Stumptner, *University of South Australia*. January, 2005. 1-920-68225-2.
 Contains the papers presented at the Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005), Newcastle, NSW, Australia, January/February 2005.
- Volume 44 - ACSW Frontiers 2005**
 Edited by Rajkumar Buyya, *University of Melbourne*, Paul Coddington, *University of Adelaide*, Paul Montague, *Motorola Australia Software Centre*, Rei Safavi-Naini, *University of Wollongong*, Nicholas Sheppard, *University of Wollongong* and Andrew Wendelborn, *University of Adelaide*. January, 2005. 1-920-68226-0.
 Contains the papers presented at the Australasian Workshop on Grid Computing and e-Research (AusGrid 2005) and the Third Australasian Information Security Workshop (AISW 2005), Newcastle, NSW, Australia, January/February 2005.
- Volume 45 - Information Visualisation 2005**
 Edited by Seok-Hee Hong, *NICTA, Australia*. January, 2005. 1-920-68227-9.
 Contains the papers presented at the Asia-Pacific Symposium on Information Visualisation, APVis.au, Sydney, Australia, January 2005.
- Volume 46 - ICT in Education**
 Edited by Graham Low, *University of New South Wales, Australia*. October, 2005. 1-920-68228-7.
 Contains selected refereed papers presented at the South East Asia Regional Computer Confederation (SEARCC) 2005: ICT Building Bridges Conference, Sydney, Australia, September 2005.
- Volume 47 - Safety Critical Systems and Software 2004**
 Edited by Tony Cant, *University of Queensland*. March, 2005. 1-920-68229-5.
 Contains all papers presented at the Ninth Australian Workshop on Safety-Related Programmable Systems, (SCS2004), Brisbane, Australia, October 2004.
- Volume 48 - Computer Science 2006**
 Edited by Vladimir Estivill-Castro, *Griffith University* and Gillian Dobbie, *University of Auckland, New Zealand*. January, 2006. 1-920-68230-9.
 Contains the papers presented at the Twenty-Ninth Australasian Computer Science Conference (ACSC2006), Hobart, Tasmania, Australia, January 2006.
- Volume 49 - Database Technologies 2006**
 Edited by Gillian Dobbie, *University of Auckland, New Zealand* and James Bailey, *University of Melbourne*. January, 2006. 1-920-68231-7.
 Contains the papers presented at the Seventeenth Australasian Database Conference (ADC2006), Hobart, Tasmania, Australia, January 2006.
- Volume 50 - User Interfaces 2006**
 Edited by Wayne Piekarski, *University of South Australia*. January, 2006. 1-920-68232-5.
 Contains the papers presented at the Seventh Australasian User Interface Conference (AUI2006), Hobart, Tasmania, Australia, January 2006.
- Volume 51 - Theory of Computing 2006**
 Edited by Barry Jay, *UTS, Australia* and Joachim Gudmundsson, *NICTA, Australia*. January, 2006. 1-920-68233-3.
 Contains the papers presented at the Twelfth Computing: The Australasian Theory Symposium (CATS2006), Hobart, Tasmania, Australia, January 2006.
- Volume 52 - Computing Education 2006**
 Edited by Denise Tolhurst, *University of New South Wales, Australia* and Samuel Mann, *Otago Polytechnic, Otago, New Zealand*. January, 2006. 1-920-68234-1.
 Contains the papers presented at the Eighth Australasian Computing Education Conference (ACE2006), Hobart, Tasmania, Australia, January 2006.
- Volume 53 - Conceptual Modelling 2006**
 Edited by Markus Stumptner, *University of South Australia*, Sven Hartmann, *Massey University, New Zealand* and Yasushi Kiyoki, *Keio University, Japan*. January, 2006. 1-920-68235-X.
 Contains the papers presented at the Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Tasmania, Australia, January 2006.
- Volume 54 - ACSW Frontiers 2006**
 Edited by Rajkumar Buyya, *University of Melbourne*, Tianchi Ma, *University of Melbourne*, Rei Safavi-Naini, *University of Wollongong*, Chris Stekettee, *University of South Australia* and Willy Susilo, *University of Wollongong*. January, 2006. 1-920-68236-8.
 Contains the papers presented at the Fourth Australasian Workshop on Grid Computing and e-Research (AusGrid 2006) and the Fourth Australasian Information Security Workshop (AISW 2006), Hobart, Tasmania, Australia, January 2006.
- Volume 55 - Safety Critical Systems and Software 2005**
 Edited by Tony Cant, *University of Queensland*. Late 2005. 1-920-68237-6.
 Contains all papers presented at the 10th Australian Workshop on Safety Related Programmable Systems, August 2005, Sydney, Australia.
- Volume 56 - Visual Information Processing 2005**
 Edited by Hong Yan, *City University of Hong Kong*, Jesse Jin, *University of Newcastle, Australia*, Zhi-qiang Liu, *City University of Hong Kong* and Daniel Yeung, *Hong Kong Polytechnic University*. Late 2005. 1-920-68238-4.
 Contains papers from the Asia-Pacific Workshop on Visual Information Processing (VIP2005), Hong Kong, December 2005.
- Volume 57 - Multimodal User Interaction 2005**
 Edited by Fang Chen and Julien Epps, *National ICT Australia*. December, 2005. 1-920-68239-2.
 Contains the proceedings of the Multimodal User Interaction Workshop 2005, NICTA-HCSNet, Sydney, Australia, 13-14 September 2005.
- Volume 58 - Advances in Ontologies 2005**
 Edited by Thomas Meyer, *National ICT Australia, Sydney* and Mehmet Orgun, *Macquarie University*. December, 2005. 1-920-68240-6.
 Contains the proceedings of the Australasian Ontology Workshop (AOW 2005), Sydney, Australia, 6 December 2005.